M. JITARU C. MACARIE St. NICULESCU

# ÎNDRUMĂTOR DE LIMBAJE DE PROGRAMARE

Scanned with OKEN Scanner

Carmen Andrew

MIHAI JITARU CASIMIR MACARIE STELIAN NICULESCU

# ÎNDRUMĂTOR DE LIMBAJE DE PROGRAMARE

EDITURA TEHNICĂ
București—1978

Scanned with OKEN Scanner

Lucrarea evidenţiază o serie de aspecte privind limbajele de programare şi utilizarea acestora în programare. Primele două capitole au un caracter general, de sinteză, prezentîndu-se caracteristicile funcţionale şi tehnice ale limbajelor de programare (structuri de programe, structuri de control, structuri de date, principiile programării structurate etc.). Capitolele următoare (3—9) sînt consacrate prezentării principalelor caracteristici ale unor limbaje de programare frecvent utilizate (FORTRAN, ALGOL, COBOL, ASSIRIS, BASIC, LPS, MAGIRIS etc.). Ultimele două capitole se referă la utilizarea combinată în programare a limbajelor ASSIRIS, COBOL, FORTRAN, la testarea şi depanarea programelor.

Lucrarea se adresează specialiştilor din unităţile de informatică şi din învăţămînt, elevilor, studenţilor, tuturor celor interesaţi să-şi diversifice cunoştinţele legate de limbajele de programare şi utilizarea acestora.

Control ştiinţific : Conf. dr. **Ion Văduva**
Redactor : **Valentina Creţu**
Tehnoredactor : **Elena Creon**
Coperta : **Alexandru Banu**

Bun de tipar : 4.09.1978. Coli de tipar : 21,75.
Tiraj : 37 500+65 exemplare legate. C.Z. 681.142.2.

Întreprinderea Poligrafică „Banat" Timişoara,
Calea Aradului nr. 1
Comanda nr. 59.

# PREFAȚA

*Dezvoltarea impetuoasă a țării noastre în cadrul procesului de făurire a societății socialiste multilateral dezvoltate implică sarcini deosebite pentru toate domeniile de activitate.*

*Informatica, alături de electronică, automatică, cibernetică etc., este chemată să contribuie la realizarea unei economii moderne, de înalt nivel tehnic, de mare productivitate și eficiență. Aceasta presupune însă ridicarea continuă a nivelului de pregătire profesională a celor ce lucrează în acest domeniu, utilizarea eficientă a calculatoarelor electronice.*

*Lucrarea își propune să prezinte, într-o manieră unitară, principalele caracteristici ale unor limbaje de programare cu o arie de răspîndire mai mare. Se urmărește o orientare a specialiștilor din informatică într-un domeniu dinamic — cel al limbajelor de programare — și, totodată, realizarea unui ghid prețios în activitatea de programare. Considerăm că însușirea principalelor caracteristici ale unui număr cît mai mare de limbaje de programare va duce la creșterea eficienței lucrărilor de programare, prin utilizarea, în soluționarea diverselor probleme, a limbajelor de programare adecvate. De asemenea, lucrarea urmărește completarea literaturii de specialitate din țara noastră, fiind prezentate unele aspecte legate de utilizarea combinată a mai multor limbaje în realizarea unor lucrări complexe, precum și prezentarea problemelor testării și punerii la punct a programelor.*

*Lucrarea se adresează specialiștilor din informatică și din învățămînt, elevilor, studenților, tuturor celor care doresc să-și îmbogățească cunoștințele despre limbajele de programare și utilizarea lor.*

*Deoarece îndrumătorul de față reprezintă un început de acest gen în literatura de specialitate din țara noastră, orice sugestie a cititorului, privind conținutul lucrării, o considerăm binevenită.*

**AUTORII**

Scanned with OKEN Scanner

# CUPRINS


Prefață ..... 5

1. Noțiuni introductive ..... 7
   1.1. Limbaje de programare ..... 7
   1.2. Clase de limbaje de programare ..... 9
   1.3. Alegerea unui limbaj de programare ..... 11
2. Caracteristici tehnice ale limbajelor de programare ..... 14
   2.1. Introducere ..... 14
   2.2. Noțiunea de algoritm ..... 15
   2.3. Elemente de teoria limbajelor formale ..... 15
   2.4. Definirea limbajelor de programare ..... 18
   2.5. Structuri de programe ..... 23
   2.6. Structuri de date în limbajele de programare ..... 30
   2.7. Structuri de control în limbajele de programare ..... 45
3. Limbaje de asamblare ..... 58
   3.1. Limbajul ASSIRIS ..... 60
   3.2. Limbajul FAST ..... 68
   3.3. Limbajul LAF-32 ..... 70
4. Limbaje pentru rezolvarea problemelor tehnico-științifice ..... 72
   4.1. Limbajul FORTRAN ..... 72
   4.2. Limbajul ALGOL ..... 95
   4.3. Compilatorul FLAG ..... 113
   4.4. Limbajul LISP ..... 115
5. Limbaje pentru probleme de gestiune ..... 125
   5.1. Limbajul COBOL ..... 125
   5.2. Limbajul LPG ..... 178
   5.3. Limbajul RPG ..... 207
6. Limbajul PL/I ..... 208
7. Limbaje de macrogenerare ..... 218
   7.1. Limbajul MAGIRIS ..... 218
   7.2. Limbajul LPS ..... 228
8. Limbaje conversaționale ..... 238
   8.1. Limbajul BASIC ..... 238
   8.2. Limbajul APL ..... 246
   8.3. Limbajul TTPL ..... 254
9. Limbaje specializate ..... 257
   9.1. Limbaje de simulare ..... 257
   9.2. Limbaje de interogare ..... 260
10. Utilizarea combinată a limbajelor ASSIRIS, COBOL, FORTRAN ..... 261
   10.1. Segmentarea programelor ..... 261
   10.2. Utilizarea combinată a subprogramelor ASSIRIS, COBOL, FORTRAN 276
11. Testarea și punerea la punct a programelor ..... 285
   11.1. Generalități ..... 285
   11.2. Tipuri de erori ..... 287
   11.3. Erori semnalate în fazele de pregătire a programelor pentru execuție .... 288
   11.4. Erori semnalate în faza de execuție a programelor ..... 303
   11.5. Facilități de testare și de punere la punct a programelor ..... 312
   11.6. Localizarea și corectarea erorilor semnalate la execuția programelor 339

*Bibliografie* ..... 344




Scanned with OKEN Scanner

# 1

# NOȚIUNI INTRODUCTIVE

## 1.1. LIMBAJE DE PROGRAMARE

În sensul cel mai larg, prin *limbaj de programare* se înțelege orice limbaj folosit pentru descrierea algoritmilor și a structurilor de date. Cel mai adesea, această noțiune se leagă de implementarea limbajului pe un calculator. La ora actuală există în lume sute de limbaje de programare definite și implementate. Încă din anul 1969, J. E. Sammet [73] dă o listă exhaustivă a 120 limbaje de programare foarte răspîndite, iar în 1972 această cifră se ridica la 170 [67]. În afară de acestea există încă multe limbaje de programare, neînregistrate, dar care se utilizează curent avînd o arie de răspîndire mai mică sau mai mare. În evoluția acestui concept [67] se marchează cîteva etape importante. Astfel, în perioada 1952—1956, considerată ca o etapă de debut, sînt definite conceptele de *programare automată*, *limbaj de asamblare* (ca un tip rudimentar de limbaj de programare) și *limbaj de programare de nivel înalt*. În această perioadă se definește și prima variantă a limbajului FORTRAN. Anii de cotitură sînt 1956 și 1957, cînd au loc cîteva întîlniri internaționale dedicate problemei limbajelor de programare, iar anii 1958—1959 sînt considerați a fi cei mai prolifici ani în istoria limbajelor de programare prin următoarele evenimente:

1) publicarea raportului privind limbajul IAL (International Algebraic Language), limbaj care va rămîne cunoscut ca ALGOL 58;

2) prezentarea formalismului BNF (Backus Naur Form) pentru descrierea sintaxei limbajului ALGOL de către J. Backus;

3) formarea unui comitet CODASYL (Conference on Data Systems Languages) pentru limbajul COBOL;

4) definirea limbajului LISP (*List processing*) și prima implementare a limbajului COMIT;

5) dezvoltarea mai multor limbaje specializate (de exemplu: DYNAMO).

Perioada 1960—1970 este considerată drept o perioadă de maturizare a limbajelor de programare, cînd paralel cu definirea de noi limbaje (COBOL, PL/I, ALGOL 68, APL, BASIC, SNOBOL, GPSS, SIMSCRIPT etc.) se realizează standarde oficiale pentru acestea, se dezvoltă proble-



Scanned with OKEN Scanner

mele teoretice ale limbajelor şi ale construirii compilatoarelor. Astăzi nu s-ar putea spune că ritmul realizării de noi limbaje a scăzut, ci că el continuă cu un şi mai mare avînt. În permanenţă sînt anunţate noi limbaje de programare. În acelaşi timp au loc numeroase dezbateri asupra unor aspecte controversate ale limbajelor, cele mai multe referiri făcîndu-se la limbajele PL/I, ALGOL 68, APL, deşi majoritatea programelor se scriu în limbajele FORTRAN, COBOL etc. Punctele de vedere se concentrează în trei direcţii, fiind susţinute de teoreticienii limbajelor de programare, de utilizatorii limbajelor şi de cei care construiesc translatoarele de limbaje de programare, fiecare dintre aceştia fiind preocupat de probleme specifice.

O problemă care tinde însă să devină generală în domeniul limbajelor de programare este cea a verificării, a demonstrării corectitudinii programelor. Aceasta este direct legată de limbajele de programare, deoarece multe din tehnicile sugerate pentru rezolvarea ei implică schimbări sau completări ale limbajelor, inserarea unor enunţuri complet diferite faţă de cele existente în programe sau chiar schimbarea metodelor de programare utilizate pînă acum. În lucrarea de faţă punctul de vedere adoptat este de a prezenta mai mult probleme specifice cu care este confruntat utilizatorul de limbaje de programare.

Mulţi programatori îşi vor pune următoarea întrebare: de ce este necesar studiul mai multor limbaje de programare cînd pentru rezolvarea problemelor curente sînt suficiente limbajele COBOL, FORTRAN şi eventual un limbaj de asamblare? În continuare se vor prezenta pe scurt o serie de argumente în favoarea unui astfel de studiu.

1. *Înţelegerea mai profundă a limbajului utilizat în mod curent.* Adesea limbajele conţin foarte multe facilităţi care, dacă sînt utilizate corect, se dovedesc de un real folos în programare, neutilizarea lor putînd conduce la un consum mare de timp calculator sau la erori logice greu de depistat. Utilizarea corectă a acestor facilităţi rezultă, în multe cazuri, mai bine din studiul lor şi în cadrul altor limbaje.

2. *Îmbogăţirea cunoştinţelor de programare cu noi topici fundamentale.* Astfel de elemente pot fi următoarele: variabile, tablouri şi tehnici de acces bazate pe utilizarea indicilor, liste şi prelucrări de liste, subrutine, proceduri, corutine, structuri de control, structuri de date etc. Modul în care acestea au fost incluse în diverse limbaje de programare scoate în evidenţă şi mai mult proprietăţile lor specifice. Se poate obţine astfel o diversificare a metodelor de programare.

3. *Evaluarea influenţei pe care o au aplicaţiile, utilizatorii şi tehnicile de implementare asupra limbajelor de programare.* De exemplu, în deter-



Scanned with OKEN Scanner

minarea influenței aplicațiilor asupra limbajelor și a evoluției acestora se pot considera mai mulți factori. Doi din cei mai importanți sînt: *structurile de date* și *structurile de programe*. Structurile de date cuprind tipurile de date elementare, relațiile dintre aceste date și metodele de acces la ele, în timp ce structurile de programe includ elementele de bază folosite în construirea programelor, relațiile dintre ele, metode de construire a programelor și segmentarea programelor.

Dacă se consideră, de exemplu, aplicațiile tehnico-științifice, structurile de date tipice sînt tablourile (mulțimi ordonate de elemente de același tip ce pot fi individualizate prin indici), iar structura de control ce se pretează a fi utilizată cel mai bine este structura repetitivă (bucla sau ciclul). La rîndul lor, aplicațiile de gestiune pun în evidență structuri de date mai complexe decît structurile matematice uzuale. Datele elementare au o varietate mai mare de tipuri: numere, șiruri de caractere etc., ele corespunzînd unor nume, adrese, cantități, prețuri etc. În același timp și structurile de date implicate sînt mai puțin omogene. Structura de bază folosită în acest caz este articolul (colecție de date de diverse tipuri), tipul de prelucrare fiind determinat de structura articolului.

4. *Alegerea mai judicioasă a limbajului de programare.* Cunoașterea caracteristicilor mai multor limbaje permite alegerea limbajului cel mai adecvat pentru un anumit proiect de programare, ceea ce poate conduce la reducerea substanțială a efortului de programare.

5. *Învățarea mai ușoară a unui nou limbaj.*

6. *Posibilitatea de a defini mai ușor un nou limbaj de programare.* Puțini programatori gîndesc ca proiectanți de limbaje. Formatul datelor de intrare pentru orice program definește în fiecare caz un limbaj, care deși în multe situații este un limbaj banal, în cazul unor proiecte mari poate constitui un limbaj de programare veritabil.

7. *Posibilitatea scrierii de programe în mai multe limbaje.* În anumite cazuri se pot elabora unele părți din programe în alte limbaje decît cel folosit cu preponderență. Acest lucru este dictat de natura problemei și de posibilitățile limbajului folosit curent. Se pot obține astfel programe mai performante.

## 1.2. CLASE DE LIMBAJE DE PROGRAMARE

J. E. Sammet, încercînd o clasificare a limbajelor de programare [73], introduce următoarele clase de limbaje: limbaje procedurale, limbaje neprocedurale, limbaje orientate pe problemă/pe aplicație, limbaje



Scanned with OKEN Scanner

speciale, limbaje pentru definirea, descrierea sau rezolvarea problemelor etc. Se precizează de la bun început că definirea unor astfel de clase nu determină gruparea limbajelor existente în mulțimi disjuncte; un limbaj poate să facă parte din una sau mai multe astfel de clase în funcție de caracteristicile acestuia.

1.2.1. *Limbaje procedurale* (sau limbaje de nivel înalt) sînt limbaje folosite pentru a descrie algoritmi; elementul cheie este faptul că se descriu atît operațiile cît și ordinea în care acestea se execută. Algoritmul astfel reprezentat constă dintr-un grup de enunțuri (instrucțiuni) ordonate. Limbajele din această clasă au o considerabilă flexibilitate, cerînd o specializare în programare. Exemple de astfel de limbaje sînt: FORTRAN, COBOL, ALGOL, PL/1 etc.

1.2.2. *Limbaje neprocedurale* (sau limbaje de nivel foarte înalt). Principala caracteristică a acestor limbaje constă în faptul că, în general, permit utilizatorilor să precizeze mai degrabă problema decît metodele de rezolvare. Prin urmare, într-un sens mai larg, se poate spune că un program neprocedural este o prescripție de rezolvare a unei probleme fără a da detalii asupra modului cum trebuie rezolvată. În general, nu este posibil să se afirme precis, dacă un anumit limbaj este neprocedural în sens absolut, dar se poate spune dacă acesta posedă sau nu anumite trăsături de neproceduralitate. Pentru aceasta trebuie să se definească ce caracteristici au limbajele neprocedurale. B. M. Leavenworth și J. E. Sammet [36] discută cinci astfel de caracteristici și anume:

1) *Referiri asociative*, adică operații de acces la date pe baza unor proprietăți intrinseci ale acestora și nu prin specificarea explicită a modului de acces sau prin programarea unui algoritm de căutare specific unor date cu o anumită structură.

2) *Operatori generali*, prin care se pot executa operații cu structuri de date de diverse tipuri; se pot astfel evita programări de cicluri pentru a realiza unele operații. Așa sînt, de exemplu, operatorii definiți în limbajul APL.

3) *Eliminarea înlănțuirilor arbitrare.* O înlănțuire arbitrară (arbitrary sequencing) este definită ca orice înlănțuire care nu este dictată de relațiile dintre datele asociate unei aplicații. Înlănțuirile dintre operații trebuie să fie strict determinate deci de relațiile dintre date, relații ce sînt explicitate în acest caz prin structura operand-operator a programelor. Astfel, această trăsătură impune eliminarea totală a operațiilor de atribuire și de salt (gen GO TO).

4) *Programare nedeterministă și procese paralele.* Ideea programării nedeterministe a fost preluată de la modul de funcționare a așa-numitelor mașini nedeterministe introduse în teoria calculabilității. O astfel de



Scanned with OKEN Scanner

mașină conține în diagrama sa de flux elemente de selecție cu o intrare și mai multe ieșiri, ieșirile fiind selectate arbitrar pentru continuarea procesului de calcul. Evident, există o strînsă legătură cu programarea paralelă, deoarece procesele de calcul pe fiecare din ieșirile selectate pot fi executate în paralel.

5) *Structuri dirijate de modele*, care au la bază principiul egalizării modelelor (pattern matching). Căutarea într-o bază de date a unor elemente pe baza unui model dat, apelarea dirijată de un model a unor proceduri de calcul sînt exemple de astfel de structuri.

1.2.3. *Limbajele specializate*, la rîndul lor, constituie o clasă foarte largă. Acestea sînt folosite pentru a realiza operații specifice unor tipuri de aplicații. În mod obișnuit, ele constau dintr-o mulțime de funcții care pot fi referite explicit; de exemplu, funcții pentru rezolvarea de sisteme de ecuații algebrice sau diferențiale, integrări numerice, operații cu mulțimi, simulări de procese continue sau discrete etc. În general, datorită structurii lor, aceste limbaje au un caracter neprocedural.

## 1.3. ALEGEREA UNUI LIMBAJ DE PROGRAMARE

Persoanele care au sarcina de a alege un limbaj adecvat pentru soluționarea unei probleme date trebuie, în general, să fie capabile să evalueze posibilitățile și limitele lui. Într-o anumită măsură, rezolvarea acestei probleme depinde de tipul aplicației date, de dimensiunea estimată a programelor, de utilizările lor posibile etc.

Pentru mulți programatori utilizarea limbajului de asamblare constituie soluția cea mai bună. Dar caracteristicile acestui limbaj depind direct de cele ale calculatorului și astfel programele scrise nu pot fi transferate pe un alt tip de calculator decît după o rescriere completă. Totuși, folosirea unui astfel de limbaj se dovedește de multe ori o soluție eficientă față de cazul programării într-un limbaj de înalt nivel, deoarece se pot utiliza in extenso toate facilitățile calculatorului.

În cazul în care decizia este de a nu utiliza limbajul de asamblare, nu există nici o metodă bine pusă la punct pentru a alege cel mai potrivit limbaj de programare pentru o problemă dată, deși au fost consacrate multe lucrări metodelor de evaluare a limbajelor și a compilatoarelor corespunzătoare. J. E. Sammet [73] enumeră totuși șapte factori care pot fi luați în considerație într-o astfel de problemă.

1. *Măsura în care limbajul este convenabil atît pentru clasa problemelor de rezolvat cît și pentru utilizatori.* Ideea constă în a vedea dacă

un limbaj conține elementele necesare rezolvării clasei de probleme propuse (structuri de date, operații cu structurile de date) și dacă aceste elemente sînt convenabile pentru utilizatorii limbajului (programatori cu experiență, programatori începători, specialiști în alte domenii), dată fiind diferența de nivel de pregătire existentă între aceștia. Se poate examina, de exemplu, dacă limbajul satisface aceste cerințe, mai întîi la un nivel general și, în caz afirmativ, se poate trece la un nivel mai de detaliu pentru a vedea dacă anumite caracteristici, importante într-o situație particulară, sînt disponibile.

2. *Posibilitatea de a utiliza limbajul pe un anumit calculator.* Aceasta revine la a examina dacă există posibilitatea de a executa programe scrise într-un limbaj pe un anumit calculator. Astfel, se poate întîmpla ca :

— să nu existe compilator sau interpreter ;

— să existe compilator sau interpreter, însă să nu poată fi utilizat pe calculatorul propriu din cauza configurației (de exemplu, memorie centrală insuficientă).

3. *Istoricul și evaluarea utilizărilor anterioare ale limbajului.* Se pot analiza o serie de elemente ca :

— reacția utilizatorilor anteriori ;

— punctul de vedere al acestora asupra aplicabilității în practică a limbajului ;

— eficiența implementării existente ;

— dificultăți de însușire a limbajului ;

— documentația existentă ;

— compatibilitatea cu alte limbaje.

4. *Eficiența implementării limbajului.* Se poate studia eficiența copilatoarelor existente și influența acestora asupra eficienței lucrărilor în ansamblu. Există trei moduri diferite de măsurare a eficienței :

a) *Eficiența execuției programului* care are o mare importanță în cazul unor programe ce se execută de multe ori.

b) *Eficiența translatării programului* care este importantă în cazul compilării și punerii la punct a unor programe mici, situație tipică în cazul instruirii. În acest caz este mai eficient un compilator rapid decît unul care să genereze un cod executabil optimizat.

c) *Eficiența scrierii, testării și folosirii programului* care în multe cazuri poate avea un rol important.

5. *Independența limbajului față de caracteristicile calculatorului și compilatorului.* O importantă problemă în acest sens este cea a transportabilității programelor. Un limbaj care este larg răspîndit și a cărui definire este independentă de caracteristicile unui calculator particular



Scanned with OKEN Scanner

constituie un mijloc eficace pentru realizarea de programe transportabile (succesul limbajului COBOL este, într-o anumită măsură, datorat independenței sale).

6. *Caracteristicile funcționale ale limbajelor.* Acestea se referă la acele aspecte ale limbajelor de programare care nu sînt considerate drept caracteristici tehnice și care în general nu fac parte din specificațiile limbajelor, ele incluzînd :

a) *Proprietățile generale ale limbajelor* :

*generalitate* — posibilitatea limbajului de a se aplica direct și efectiv la o clasă largă de probleme ;

*simplitate* — ușurința de a fi învățat, utilizat și implementat ;

*concizіune* — exprimarea succintă a problemelor ;

*naturalețe* — exprimarea într-o formă cît mai naturală a problemelor ;

*consistență* — aplicarea constantă a aceloraşi reguli într-un acelaşi mod în întreg limbajul ;

*exprimabilitate* — ușurința cu care se scriu programe și mai ales programe fără erori ;

*lizibilitate* — ușurința cu care se citesc programele.

b) *Scopul limbajelor.* Definind scopul limbajelor, se precizează : tipul aplicațiilor, tipul limbajelor (procedural, neprocedural, extensibil etc.), tipul utilizatorilor potențiali, tipul calculatorului etc.

c) *Standardizarea limbajului.* Standardele constituie factor cheie în definirea și utilizarea limbajului cu implicație directă asupra transportabilității programelor.

d) *Evaluarea limbajelor bazată pe utilizarea lor practică.* Aceasta permite determinarea avantajelor utilizării limbajelor în diverse cazuri concrete, dacă și-au atins sau nu obiectivele propuse inițial.

7. *Caracteristicile tehnice ale limbajelor.* În timp ce caracteristicile funcționale ale limbajelor pot preveni utilizarea lor în cazul unor aplicații particulare, un rol hotărîtor în alegerea unui limbaj îl au caracteristicile tehnice ale acestuia, importanța elementelor structurale ale sale. Cele mai multe dintre aceste elemente vor fi descrise în capitolul următor, acestea constituind un cadru pentru discutarea fiecăruia dintre limbajele ce vor fi prezentate. Astfel, vor fi discutate metode de definire a limbajelor de programare, elemente ce intră în structura programelor, structuri de date în limbajele de programare, structuri de control etc.

Scanned with OKEN Scanner

# 2

# CARACTERISTICI TEHNICE ALE LIMBAJELOR DE PROGRAMARE

## 2.1. INTRODUCERE

O problemă importantă în studiul limbajelor de programare este specificarea sau definirea acestora. În cazul unei specificări minimale este necesar să se definească :

— mulţimea simbolurilor ce pot fi folosite în programele corecte ;
— mulţimea programelor corecte ;
— semnificaţia programelor corecte.

Definirea simbolurilor permise este un lucru relativ simplu. Definirea mulţimii programelor corecte este în schimb o problemă mult mai dificilă. De obicei, se defineşte o clasă de programe permise printr-o mulţime de reguli care permit construirea programelor corecte. Cel de-al treilea aspect, definirea semnificaţiei fiecărui program corect este cel mai dificil. Una din metodele folosite în acest caz este de a defini o funcţie care asociază fiecărui program corect o frază dintr-un alt limbaj al cărei sens îl înţelegem. Din punct de vedere ştiinţific specificarea completă a unui limbaj de programare presupune definirea *sintaxei*, *semanticii* şi *pragmaticii* limbajului. Aceste trei elemente au fost introduse de C. Morris încă din 1938 [47] şi reluate în 1955 [48]. În contextul limbajelor de programare ele au fost precizate de către H. Zemanek [101], aceste trei elemente constituind cele trei dimensiuni ale semioticii limbajelor de programare. Sensul în care aceşti trei termeni vor fi utilizaţi în continuare se poate deduce din următoarele definiţii aproximative :

— *sintaxa* : studiul structurii limbajului ;
— *semantica* : studiul semnificaţiei limbajului ;
— *pragmatica* : studiul tuturor celorlalte aspecte ale limbajului (de exemplu eficienţa, relaţia cu utilizatorii etc.).

Deşi această clasificare se utilizează destul de mult, există unele situaţii în care nu se poate spune despre un anumit element al limbajului de competenţa cui este. De aceea în multe cazuri un acelaşi element este inclus de diverşi autori în categorii diferite.



Scanned with OKEN Scanner

Dacă în ceea ce privește definirea formală a sintaxei și semanticii limbajelor de programare lucrurile sînt destul de clare (existînd multe lucrări în acest sens), în ceea ce privește definirea formală a pragmaticii problema este complet deschisă. Astfel, definirea riguroasă a limbajelor de programare este încă o problemă deschisă a informaticii teoretice cu implicații asupra informaticii aplicate.

## 2.2. NOȚIUNEA DE ALGORITM

Limbajul de programare constituie, în esență, un mijloc de a reprezenta algoritmi. Un algoritm constă dintr-o secvență finită, de operații ordonată și complet definită care pornind de la date (intrări) produce rezultate (ieșiri). Proprietățile esențiale ale unui algoritm sînt:

1) să fie *bine definit*, adică operațiile cerute să fie specificate riguros și fără ambiguitate;

2) să fie *descris foarte exact* așa încît o mașină programabilă să-l poată realiza;

3) să fie *efectiv*, adică să se termine întotdeauna după efectuarea unui număr finit de operații;

4) să fie *universal*, adică să permită rezolvarea unei clase de probleme.

## 2.3. ELEMENTE DE TEORIA LIMBAJELOR FORMALE

Se numește *vocabular* $V$ o mulțime finită de elemente numite simboluri (uneori cuvinte). O *frază* este o secvență finită de simboluri din $V$. În cele ce urmează elementele lui $V$ se vor nota cu litere mici $a$, $b$, $c$, $d$, iar frazele cu $t$, $u$, $v$, $x$, $y$, $z$. Există o frază notată cu $e$, numită *frază vidă*, care nu conține nici un simbol.

Formal, fiind dat un vocabular $V$, frazele pe $V$ se pot defini recursiv astfel:

1) $e$ este o frază pe $V$;

2) dacă $x$ este o frază pe $V$ și $a$ un element din $V$, atunci $xa$ este o frază pe $V$;

3) $y$ este o frază pe $V$ dacă și numai dacă ea rezultă după regulile 1) și 2).

Dacă $x$ și $y$ sînt fraze pe un vocabular $V$, atunci fraza $xy$ obținută prin juxtapunerea frazelor $x$ și $y$ este numită *concatenarea* lui $x$ cu $y$.

De exemplu, dacă $x=abc$ și $y=cd$, atunci $xy=abccd$. Pentru orice frază $x$, $xe=ex=x$.

*Lungimea* unei fraze $x$, notată $|x|$, este numărul simbolurilor din frază. De exemplu, dacă $x=a_1a_2\ldots a_n$, atunci $|x|=n$. De observat că

$$|e|=0 \text{ și } |xy|=|x|+|y|.$$

Un *limbaj* $L$ pe un vocabular $V$ este o mulțime de fraze pe $V$. Dacă se notează cu $V^*$ mulțimea tuturor frazelor pe vocabularul $V$, inclusiv fraza $e$, atunci un limbaj $L$ pe $V$ este o submulțime a lui $V^*$, deci $L\subseteq V^*$.

Dacă, de exemplu, $V=\{a, b\}$, atunci $V^*=\bigcup_{i\in N} S_i(V)$, unde $N$ este mulțimea numerelor întregi nenegative, iar $S_i(V)$ este mulțimea frazelor de lungime $i$ pe $V$, adică

$$S_0(V)=\{e\},\ S_1(V)=\{a, b\},\ S_2(V)=\{aa, ab, ba, bb\},\ \ldots$$

Mulțimea tuturor frazelor nevide pe $V$ se notează cu $V^+$ ; se remarcă ușor că $V^*=V^+\bigcup\{e\}$.

Dacă un limbaj $L$ este o submulțime a lui $V^*$, problema care se pune este cum se poate reprezenta $L$, mai ales în cazul în care $L$ este infinit. Evident, dacă $L$ constă dintr-un număr finit de fraze, atunci listarea tuturor frazelor este cel mai simplu mijloc de a-l reprezenta.

În cazul în care $L$ este infinit, o metodă des utilizată pentru a-l reprezenta este sistemul generator, numit *gramatică*. În acest caz, orice frază din $L$ se poate construi prin metode bine definite, folosind *regulile gramaticii* (obișnuit numit *producții*).

În continuare se vor face referiri doar la o clasă de gramatici, numite *gramatici Chomsky*, după numele celui care le-a propus.

O gramatică $G$ pentru un limbaj $L$ folosește două mulțimi finite și disjuncte de simboluri și anume :

— o mulțime $V_T$, de *simboluri terminale*, care este chiar vocabularul pe care $L$ este definit ;

— o mulțime $V_N$, de *simboluri neterminale*, care vor fi folosite în procesul de generare a frazelor limbajului.

Elementul principal al unei gramatici este o mulțime finită $P$, de *reguli de formare*, sau *producții*, care descrie modul în care se generează frazele limbajului.

O *producție* este o pereche ordonată de fraze $(\alpha, \beta)$ cu $\alpha\in(V_N\bigcup V_T)^* V_N (V_N\bigcup V_T)^*$, deci o frază ce conține cel puțin un simbol neterminal, iar $\beta\in(V_N\bigcup V_T)^*$, deci orice frază din *vocabularul total* al limbajului ($V_N\bigcup V_T$ este numit vocabular total). În acest caz se mai spune că fraza $\beta$ derivă din fraza $\alpha$ prin gramatica $G$, din care cauză o producție $(\alpha, \beta)$ se mai notează și $\alpha\to\beta$.



Scanned with OKEN Scanner

Limbajul definit de gramatică este mulțimea tuturor frazelor formate numai din simboluri terminale și care pot fi derivate plecînd cu o frază particulară formată dintr-un simbol neterminal special, notat de obicei cu $S$ și numit *simbol de start*, sau *axioma gramaticii*. Dacă, de exemplu, $S$ este neterminalul „*frază*", atunci regulile arată cum se construiesc toate frazele limbajului.

Ținînd seama de cele prezentate, se poate da definiția formală a gramaticii.

O *gramatică* $G$ este un grup de patru elemente $(V_N, V_T, P, S)$, unde:

(1) $V_N$ este o mulțime finită de simboluri neterminale, numite uneori și categorii sintactice;

(2) $V_T$ este o mulțime finită de simboluri terminale cu $V_N \cap V_T = \emptyset$;

(3) $P$ este o submulțime finită a produsului cartezian

$$(V_N \cup V_T)^* V_N (V_N \cup V_T)^* \times (V_N \cup V_T)^*,$$

ale cărui elemente $(\alpha, \beta)$ sînt numite producții, iar $V_N \cup V_T$ constituie vocabularul total;

(4) $S$ este un simbol special din $V_N$, numit axioma gramaticii.

Un exemplu de gramatică este $G=(\{S, X, Y\}, \{a, b, c, d\}, P, S)$, unde:

(1) $V_N=\{S, X, Y\}$, cu $S$, $X$, $Y$ simboluri neterminale;

(2) $V_T=\{a, b, c, d\}$, cu $a$, $b$, $c$, $d$ simboluri terminale;

(3) $P=\{S \to XY,\ X \to a,\ X \to b,\ Y \to c,\ Y \to d\}$ fiind mulțimea regulilor gramaticii;

(4) $S \in V_N$ fiind axioma gramaticii.

Fie o gramatică $G=(V_N, V_T, P, S)$. O pereche $(\alpha, \beta)$ se numește *derivare simplă* în gramatica $G$ și se notează $\alpha \Rightarrow \beta$, dacă există o regulă $\gamma \to \delta$ din $P$ și două fraze $\alpha_1$, $\alpha_2$ din $(V_N \cup V_T)^*$, astfel încît $\alpha=\alpha_1\gamma\alpha_2$ și $\beta=\alpha_1\delta\alpha_2$.

Se numește *derivare de lungime* $k$ a lui $\beta$ din $\alpha$ în gramatica $G$ și se notează $\alpha \overset{k}{\Longrightarrow} \beta$, un șir de derivații simple $\alpha_0 \Rightarrow \alpha_1$, $\alpha_1 \Rightarrow \alpha_2$, ..., $\alpha_{k-1} \Rightarrow \alpha_k$, unde $\alpha_0=\alpha$ și $\alpha_k=\beta$.

Dacă $\alpha \overset{k}{\Longrightarrow} \beta$ pentru orice $k \geq 0$, atunci se va folosi notația $\alpha \overset{*}{\Longrightarrow} \beta$.

În aceste condiții se poate spune că limbajul generat de gramatica $G$, notat de obicei $L(G)$, este mulțimea tuturor frazelor $x \in V_T^*$ astfel încît $S \overset{*}{\Longrightarrow} x$ sau $L(G)=\{x \mid S \overset{*}{\Longrightarrow} x \text{ și } x \in V_T^*\}$.

**Exemplu.** Fie gramatica $G=(\{S\}, \{a, b\}, P, S)$, unde $P$ constă din producțiile

$$S \to aSb,\ S \to ab,$$

și fie următoarea derivare:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb.$$

În primul pas s-a aplicat prima regulă din $P$, $S$ fiind înlocuit cu $aSb$, apoi s-a aplicat din nou prima regulă și în pasul al treilea regula a doua din $P$, $S$ fiind înlocuit cu $ab$. Se poate spune că $S \overset{3}{\Longrightarrow} aaabbb$ sau $S \overset{3}{\Longrightarrow} a^3b^3$ și că fraza $a^3b^3 \in L(G)$. Se poate arăta că

$$L(G)=\{a^n b^n \mid n > 1\}.$$



Scanned with OKEN Scanner

Dacă se impun anumite restricţii asupra formei producţiilor, se poate obţine o clasificare a gramaticilor.

Fie o gramatică $G=(V_N, V_T, P, S)$. Se spune că $G$ este de tip $i$, $0 \leqslant i \leqslant 3$, dacă şi numai dacă $P$ satisface una din restricţiile :

0. Nici o restricţie.

1. Fiecare producţie din $P$ este de forma $\alpha \rightarrow \beta$, unde $|\alpha| \leqslant |\beta|$, adică lungimea frazei $\alpha$ este mai mică sau egală decît lungimea frazei $\beta$ şi $\alpha, \beta \in (V_N \cup V_T)^*$.

2. Fiecare producţie din $P$ este de forma $A \rightarrow \alpha$, unde $A$ este un neterminal, deci $A \in V_N$ şi $\alpha \in (V_N \cup V_T)^*$.

3. Fiecare producţie din $P$ este de forma $A \rightarrow xB$ sau $A \rightarrow x$, unde $A, B \in V_N$ (deci simboluri neterminale), iar $x$ este o frază din $V_T^*$.

Gramaticile de tip 0 sînt numite şi *gramatici fără restricţii*, gramaticile de tip 1 *gramatici dependente de context*, cele de tip 2 *gramatici independente de context* şi cele de tip 3, care sînt cele mai restrictive, *gramatici liniare-dreapta* sau *gramatici regulate*.

Această clasificare este dată de N. Chomsky, fiind întîlnită şi sub denumirea de *ierarhie Chomsky*. La rîndul lor, limbajele generate de o gramatică de tip $i$ poartă denumirea de *limbaje de tip* $i$ ; astfel există :

0. Limbaje de tip 0 ;
1. Limbaje de tip 1 sau limbaje dependente de context ;
2. Limbaje de tip 2 sau limbaje independente de context ;
3. Limbaje de tip 3 sau limbaje liniar-dreapta sau limbaje regulate.

Ierarhia stabilită între gramatici se păstrează şi în cazul limbajelor.

## 2.4. DEFINIREA LIMBAJELOR DE PROGRAMARE

Conceptul de gramatică, introdus în paragraful anterior, constituie un model foarte bun pentru definirea limbajelor de programare. Astfel, dacă se consideră un vocabular ce conţine elementele cu care se vor scrie programe, atunci se poate considera că programele sînt fraze corecte pe acest vocabular. Limbajul de programare este deci o mulţime de programe, iar gramatica limbajului este o mulţime de reguli care vor determina frazele ce aparţin limbajului. Dintre tipurile de limbaje formale din ierarhia Chomsky, ca model pentru limbajele de programare s-au folosit cel mai mult limbajele independente de context. În cazul unui limbaj independent de context substituirea unei fraze este independentă de contextul în care se află aceasta ; acest fapt simplifică considerabil studiul acestui tip de gramatică.

În cele ce urmează se va prezenta modul de definire a sintaxei limbajelor de programare şi se vor face scurte referiri la modul de definire a semanticii.



Scanned with OKEN Scanner

2.4.1. **Definirea sintaxei limbajelor de programare.** Deoarece o limbă naturală este în general ambiguă şi redondantă, pentru definirea limbajelor de programare este necesară introducerea unei metode formale riguroase. Deci, este necesară utilizarea unui limbaj formal, cu ajutorul căruia să se exprime gramatica limbajului. Un astfel de limbaj este numit *metalimbaj*.

Prima contribuţie în această direcţie aparţine lui J. Backus, concretizată în *metalimbajul BNF* generat de o gramatică BNF (Backus Naur Form). Acest metalimbaj a fost folosit pentru prima dată la definirea sintaxei limbajului ALGOL 60 [52]. Un alt metalimbaj cu o largă răspîndire este cel folosit în definirea limbajului COBOL, denumit *metalimbaj CBL* (COBOL-like) generat de o gramatică CBL.

**Metalimbajul** *BNF*. O gramatică BNF este o gramatică independentă de context, fiind compusă dintr-o mulţime finită de reguli cu ajutorul cărora se alcătuiesc fraze ce descriu sintaxa unui limbaj. Cum sintaxa se referă doar la forma frazelor limbajului, fără nici o referire la sensul acestora, rezultă că prin metalimbajul BNF se specifică programele corecte din punct de vedere sintactic.

Vocabularul terminal $V_T$ al unei gramatici BNF conţine simbolurile $::=$, $|$, $<$, $>$ (metasimboluri) la care se adaugă o mulţime finită de elemente numite *metaconstante*. Metaconstantele sînt elemente din vocabularul terminal al limbajului care se defineşte cu această gramatică BNF, care se iau aşa cum sînt fără nici o modificare.

Vocabularul neterminal $V_N$ conţine o mulţime finită de simboluri neterminale, numite *metavariabile*. O metavariabilă se obţine dintr-un şir de caractere incluse între metasimbolurile $<$ şi $>$; de exemplu, $<$program$>$ este o metavariabilă (una dintre metavariabile este şi axioma S a gramaticii BNF).

O *metaexpresie* se obţine din metavariabile şi metaconstante unite prin operatorul de concatenare şi/sau metaoperatorul $|$ (care se citeşte *sau* indicînd o nouă alternativă în definiţie).

O frază din metalimbajul BNF are forma generală:

*metavariabilă* $::=$ *metaexpresie*

şi se citeşte „metavariabilă *este definită prin* metaexpresie". Forma acestor fraze este, într-un anumit sens, similară cu cea a instrucţiunilor de atribuire din limbajele de programare.

**Exemple. 1.** $<$ *cifră binară* $>::=0 \mid 1$, ceea ce înseamnă că metavariabila $<$ *cifră binară* $>$ este definită sau prin 0 sau prin 1.

**2.** $<$ *cifră* $>::=0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$.

**3.** $<$ *nr. întreg* $>::=<$ *cifră* $> \mid <$ *nr. întreg* $><$ *cifră* $>$.
Această frază în limbaj BNF exprimă faptul că un număr întreg se defineşte ca fiind o cifră sau un număr întreg urmat de o cifră. Această definiţie implică o recursivitate.



Scanned with OKEN Scanner

ceea ce înseamnă că se pot construi numere întregi cu oricît de multe cifre (în limbajele de programare se impune de obicei o limită).

Se poate face acum o legătură între definirea formală a unei gramatici independente de context și metalimbajul BNF. Pentru aceasta se introduce noțiunea de *text în metalimbajul BNF* ca fiind o succesiune de fraze BNF. Se afirmă că pentru o gramatică independentă de context $G=(V_N, V_T, P, S)$ există un text BNF care definește complet gramatica $G$. Acest lucru se poate ușor verifica dacă se demonstrează că atît textul BNF cît și gramatica $G$ generează același limbaj.

Fie, de exemplu, următorul text BNF :

$$< A > ::= < B >,$$

$$< B > ::= a < B > a \mid b,$$

unde $A$ și $B$ sînt metavariabile, iar $a$ și $b$ metaconstante și gramatica independentă de context $G=(V_N, V_T, P, S)$, unde $V_N=\{A, B\}$, $V_T=\{a, b\}$, $S=A$, $P=\{A \to B.\ B \to aBa, B \to b\}$. Se poate ușor verifica că atît textul BNF dat cît și gramatica $G$ generează limbajul

$$L(G)=\{b, aba, aabaa, \ldots\}=\{a^n b a^n \mid n \geqslant 0\},$$

unde prin $a^n$ s-a notat repetarea de $n$ ori a lui $a$, iar $a^0=e$. Limbajul $L(G)$ este un limbaj infinit, ceea ce este asigurat în textul BNF prin fraza care definește pe $< B >$ și care folosește în definire tot pe $< B >$.

Observație. În general, dacă o gramatică $G$ conține reguli de forma $A \to \alpha A \beta$ unde $A$ este un neterminal, se spune că $G$ este *recursivă* în raport cu $A$. Dacă regulile sînt de forma $A \to A\beta$ sau $A \to \alpha A$, atunci $G$ este *recursivă la stînga*, respectiv *recursivă la dreapta* în raport cu $A$. Prin urmare, recursivitatea este o proprietate generală a gramaticilor independente de context.

*Metalimbajul* CBL. În ciuda simplității sale, metalimbajul BNF se dovedește de multe ori a fi o metodă nepractică pentru prezentarea sintaxei limbajelor de programare, în special cînd aceasta este destinată direct programatorilor. Astfel, el nu poate fi utilizat eficient mai ales în cazul unor construcții sintactice ce conțin elemente opționale, elemente ce se repetă de mai multe ori, elemente la alegere, situații în care indicarea tuturor alternativelor conduce la metaexpresii laborioase, greu de urmărit. De exemplu, pentru definirea unui număr întreg cu semn se folosesc de obicei următoarele fraze BNF :

$$< nr.\ întreg\ cu\ semn > ::= + < întreg > \mid - < întreg > \mid < întreg >,$$

$$< întreg > ::= < cifră > \mid < întreg > < cifră >.$$

În cazul studierii teoretice a proprietăților gramaticilor, aceste insuficiențe ale metalimbajului BNF se dovedesc totuși neimportante. Majoritatea celor ce au folosit metalimbajul BNF pentru descrierea gramaticii



Scanned with OKEN Scanner

limbajelor de programare i-au extins gramatica în diverse moduri pentru a obține o reprezentare mai naturală a regulilor. Cea mai utilizată dintre aceste extinderi este cea folosită în descrierea gramaticii limbajului COBOL, care în cele ce urmează va fi numită gramatică CBL, iar limbajul generat *metalimbaj CBL*. Așa cum este folosit însă în definirea limbajului COBOL, metalimbajul CBL nu s-a putut folosi în prezentarea întregii gramatici a acestuia. Prin intermediul unor metafraze (numite obișnuit formate) se descrie doar structura sintactică a instrucțiunilor, a anumitor rubrici și paragrafe. Metafrazele în acest caz sînt similare cu metaexpresiile din metalimbajul BNF, acestea alcătuindu-se după următoarele reguli:

1. *Metavariabilele* se compun din succesiuni de litere mici, cifre și caracterul —.

2. *Metaconstantele* sînt de două tipuri:

— cuvinte opționale din limbajul ce se definește (notate cu majuscule);

— cuvinte obligatorii din limbajul ce se definește (notate cu majuscule și subliniate).

3. *Operatorii* sînt în număr de doi: operatorul de concatenare și operatorul ... ce indică repetarea de mai multe ori a unității sintactice ce-l precede imediat.

4. *Parantezele pătrate* se folosesc pentru a indica faptul că unitățile incluse în interior sînt opționale, iar *parantezele acolade* includ în interior o listă de elemente, ordonate pe verticală, din care se va alege unu (se înlocuiește astfel operatorul | ).

Iată, de exemplu, o metafrază de acet tip ce descrie unul din formatele generale ale instrucțiunii ADD din limbajul COBOL:

$$\underline{\text{ADD}} \quad \left\{\begin{matrix} identificator\text{-}1 \\ literal\text{-}1 \end{matrix}\right\} \left[\left\{\begin{matrix} identificator\text{-}2 \\ literal\text{-}2 \end{matrix}\right\}\right] \ldots \underline{\text{GIVING}} \ identificator\text{-}n$$

$$[\underline{\text{ROUNDED}}] \ [\text{ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \ enunț\text{-}imperativ].$$

Este convenabil totuși a se considera metalimbajul CBL ca o extensie a metalimbajului BNF [60], în care metafrazele au același format general, iar metaexpresiile se alcătuiesc după regulile 1—4 prezentate mai sus, doar că metavariabilele se definesc ca în metalimbajul BNF.

**Exemple.** 1. $< enunț\ ADD > ::= \underline{\text{ADD}} \left\{\begin{matrix} < identificator > \\ < număr > \end{matrix}\right\} \left[\left\{\begin{matrix} < identificator > \\ < număr > \end{matrix}\right\}\right] \ldots$

$\underline{\text{GIVING}} < identificator > [\underline{\text{ROUNDED}}] [\text{ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} < enunț\ imperativ >]$

2. $< nr.\ întreg\ cu\ semn > ::= [\pm] < cifră > \ldots$

3. $< identificator > ::= < literă > \left[\left\{\begin{matrix} < literă > \\ < cifră > \end{matrix}\right\}\right] \ldots$

Naturaleţea şi conciziunea metalimbajului CBL astfel definit se poate uşor remarca dacă se compară modul de definire a unui număr cu semn în cele două metalimbaje (vezi exemplele de mai înainte), sau dacă se scrie o metafrază BNF pentru definirea enunţului ADD din limbaj COBOL şi se compară cu exemplul 1.

2.4.2. **Definirea semanticii limbajelor de programare.** Dacă în cazul definirii sintaxei limbajelor de programare problema principală este cea a specificării mulţimii programelor corecte, în cazul definirii semanticii problema centrală este cea a specificării semnificaţiei programelor corecte. În acest sens s-au conturat două metode.

Una din aceste două metode constă în a defini o funcţie care asociază fiecărui program corect o frază dintr-un alt limbaj, frază al cărei sens se poate înţelege. Funcţia astfel definită devine un *compilator* pentru limbajul de definit. Un astfel de limbaj, al cărui sens se poate înţelege, este, de exemplu, *calculul lambda* [14, 94]. În acest caz se poate defini sensul unui program dintr-un limbaj de programare printr-un *program echivalent* în *calculul-λ* (de fapt o *expresie-λ*). Prin program echivalent se înţelege un program ce defineşte aceeaşi funcţie.

A doua metodă constă în definirea unei *maşini de calcul idealizate* sau a unei *maşini abstracte.* În acest caz, sensul unui program se poate specifica prin efectul lui asupra acestei maşini abstracte, lansată dintr-o stare iniţială predeterminată. Maşina abstractă astfel definită devine un *interpreter* pentru limbaj. Un astfel de interpreter este de exemplu maşina abstractă utilizată pentru definirea limbajului PL/I, definită cu ajutorul metalimbajului VDL (Vienna Definition Language) [14].

Rezumînd, se poate spune că există două moduri principale de definire a sensului programelor corecte scrise într-un limbaj de programare $L_1$ :

1) *printr-un compilator, COMP*, care defineşte modul cum programele sursă, *PS*, pot fi translatate în programe echivalente *PE*, $PE = COMP(PS)$, dintr-un limbaj $L_2$ cu semantică cunoscută ;

2) *printr-un interpreter, INT*, care defineşte, pentru fiecare program $P$ din $L_1$ şi datele $D$ asociate acestuia, un algoritm pentru calcularea valorii $v(P(D))$, $v(P(D)) = INT(P, D)$, care rezultă prin execuţia programului $P$ pentru datele $D$.

Este uneori folosită o combinaţie a celor două metode de definire a semanticii limbajelor de programare [53]. Baza acestei metode este de a exprima semantica prin comportarea unei *maşini abstracte* care execută programe scrise într-o versiune idealizată a limbajului $L_1$. Definirea presupune două etape : în prima etapă are loc o translatare a programului sursă într-o formă abstractă (numit *program abstract*), iar a doua etapă constă în interpretarea programului abstract. Structura unui program abstract este definită în *sintaxa abstractă* a limbajului,



Scanned with OKEN Scanner

care pune în evidenţă doar acele elemente structurale ale programului care sînt relevante în interpretarea sa. Comparată cu sintaxa în care se scrie programul sursă (numită şi *sintaxă concretă*), sintaxa abstractă are proprietăţile :

1) este independentă de detaliile de reprezentare (setul de caractere folosit, cuvinte-cheie, reguli de punctuaţie etc.) ;

2) este independentă de o serie de reguli de ordonare (de exemplu ordonarea enunţurilor nu este relevantă ; este important doar blocul din care ele fac parte).

## 2.5. STRUCTURI DE PROGRAME

2.5.1. **Introducere.** Programele scrise într-un anumit limbaj nu sînt colecţii arbitrare de enunţuri, ci au o structură internă bine definită. În cele ce urmează un program va fi privit ca o entitate complet structurată, structura lui internă fiind numită şi *structură logică* a programului sau, simplu, *structura programului*.

Cunoaşterea structurii programelor este importantă atît pentru înţelegerea limbajelor de programare şi a programelor scrise cu ajutorul lor, cît şi pentru înţelegerea procesului de compilare sau interpretare, proces în care structura logică a programelor este analizată şi verificată de compilator sau interpreter înainte de a trece la generarea codului obiect sau interpretarea programului.

Elementele structurale din care se realizează programele, în cele mai multe dintre limbajele de programare, formează o ierarhie. Astfel de elemente, din această ierarhie, sînt următoarele : caracterul, tokenul, expresia, enunţul, enunţul compus, blocul, procedura, programul etc., numite şi *unităţi sintactice*. În unele limbaje de programare, cum sînt limbajele de tip ALGOL, structurile de programe sînt *recursive*. Unităţile structurale prezentate vor fi expuse, pe scurt, în continuare.

2.5.2. **Caracterul.** Caracterul este cea mai mică unitate structurală a unui program. În majoritatea limbajelor de programare mulţimea caracterelor este formată din caractere alfabetice, caractere numerice, caractere speciale. Alegerea mulţimii caracterelor de bază este una din primele probleme ce trebuie rezolvată atunci cînd se defineşte sintaxa unui limbaj. Există, în general, mai multe mulţimi de caractere de bază, care diferă prin caracterele speciale. În unele cazuri se utilizează mulţimi de caractere de bază nestandard, cum este cazul limbajului APL. Mulţimea caracterelor de bază folosite determină şi echipamentele de intrare/ieşire ce pot fi utilizate.



Scanned with OKEN Scanner

Privit la acest nivel, *un program este o succesiune liniară de caractere.* Pentru a reprezenta elementele structurale ale programului este necesară delimitarea, într-un anumit mod, a acestei succesiuni, marcîndu-se astfel blocurile, secţiunile, enunţurile, expresiile etc. Pentru a realiza acest lucru se folosesc aşa-numiţii *delimitatori* : separatori şi paranteze.

Un *separator* este un simbol care, inserat într-o succesiune de caractere, permite delimitarea unor unităţi sintactice ce urmează a fi tratate separat. Exemple de separatori sînt *blancul* (sau *spaţiul*), punctul, virgula etc.

*Parantezele* sînt perechi de simboluri distincte care delimitează o porţiune a unui program. Principalul avantaj al utilizării parantezelor este faptul că ele permit specificarea structurilor incluse una în alta. Se pot reprezenta astfel *structurile recursive*, acestea constituind cel mai important mijloc de a crea structuri complexe plecînd de la componente elementare. Cel mai adesea ca paranteze se folosesc simbolurile ( ,) şi [ ,]. În unele limbaje există enunţuri care acţionează ca paranteze. Astfel sînt, de exemplu, perechile **begin ... end** în ALGOL şi DO ... END în PL/I.

2.5.3. **Tokenul.** Un *token* este un element structural ce constă dintr-un caracter sau un grup de caractere. În cele mai multe limbaje se consideră drept token identificatorii, constantele, cuvintele-cheie, operatorii etc.

*Identificatorii* permit efectuarea de referiri la date, subrutine etc. Regulile după care se construiesc identificatorii sînt specifice fiecărui limbaj în parte. De exemplu, o regulă general folosită este următoarea :

$$\begin{aligned} &< \textit{identificator} > ::= < \textit{literă} > \mid \\ &\qquad < \textit{identificator} > < \textit{literă} > \mid \\ &\qquad < \textit{identificator} > < \textit{cifră} > \end{aligned}$$

*Cuvintele-cheie* sînt identificatori folosiţi ca simboluri terminale în sintaxa enunţurilor (de exemplu IF, THEN şi ELSE) şi fac parte din vocabularul de simboluri terminale al gramaticii limbajului. Dacă un cuvînt-cheie nu poate fi folosit ca identificator, atunci el este numit *cuvînt rezervat.* Introducerea cuvintelor rezervate este o consecinţă directă a modului cum se face analiza sintactică în timpul procesului de compilare. De remarcat faptul că analiza sintactică se face mai uşor în cazul definirii de cuvinte rezervate, cum este cazul limbajului COBOL. În limbajul FORTRAN noţiunea de cuvînt rezervat nu apare, însă analiza sintactică se face mai dificil.

*Operatorii* sînt în general caractere speciale sau identificatori folosiţi pentru a reprezenta operaţii primitive definite în diverse limbaje. În unele limbaje se folosesc ca operatori numai caractere speciale (de exemplu, în limbajul APL), în altele numai identificatori (de exemplu, în limbajul



Scanned with OKEN Scanner

LISP). Există și limbaje în care se folosesc ca operatori atît caractere speciale cît și identificatori (de exemplu, limbajul COBOL).

2.5.4. **Expresia.** În limbajele de programare expresiile se întîlnesc într-o mare varietate de forme. În cele mai multe cazuri expresiile se compun din termeni (variabile și constante) uniți prin operatori și grupați uneori în interiorul unor paranteze. Expresiile constituie unități sintactice de bază cu ajutorul cărora se alcătuiesc enunțuri. Totodată expresia constituie o formă de *structură recursivă* des întîlnită în limbajele de programare.

Indiferent de structura lor sintactică, există diferențe între limbaje în ceea ce privește importanța expresiilor. Astfel, în timp ce în limbaje ca FORTRAN și COBOL importanța expresiilor este relativ mică, acestea utilizîndu-se în cadrul enunțurilor cînd este necesară calcularea unor valori, în alte limbaje, cum sînt LISP și APL, expresiile constituie structuri sintactice centrale. De exemplu, în limbajul APL un program este o succesiune de expresii.

2.5.5. **Enunțul.** Enunțul (instrucțiunea) este un element structural foarte important în cadrul structurilor de programe, sintaxa și structura lui avînd un efect direct asupra regularității și lizibilității programelor. În unele limbaje toate enunțurile au același format, în timp ce în altele se utilizează o sintaxă diferită pentru fiecare tip de enunț. Prima situație scoate în evidență regularitatea limbajului (cazul limbajelor APL și SNOBOL), în timp ce a doua determină lizibilitatea limbajului, deoarece utilizarea unei varietăți largi se structuri sintactice pentru enunțuri permite exprimarea naturală a operațiilor (cazul limbajului COBOL).

Datorită importanței enunțului de multe ori *un program se definește ca o listă (sau o secvență) de enunțuri*. Enunțurile se împart de obicei în două categorii și anume : *enunțuri declarative* și *enunțuri executive*. *Enunțurile declarative* descriu datele asociate unui program și în mod obișnuit între ele nu există reguli de înlănțuire.

*Enunțurile executive* descriu *operații* de executat asupra unor *date*. Astfel, un program constă dintr-o mulțime de enunțuri a căror ordine textuală nu este în general identică cu ordonarea în timp a operațiilor corespunzătoare [96].

În cele mai multe dintre cazuri enunțurile descriu operații simple, fiind numite și *enunțuri simple*. Spre deosebire de acestea în limbajele de programare există și *enunțuri structurate* care pot include la rîndul lor alte enunțuri. Un exemplu este următorul enunț condițional din ALGOL [52] :

$$< \textit{enunț condițional} > ::= \text{IF} < \textit{expresie booleană} > \text{THEN} < \textit{enunț} > \text{ELSE} < \textit{enunț} >$$

care este un enunț structurat deoarece fiecare dintre alternative este un nou enunț. Există situații în care este util a se considera grupuri de enunțuri pentru a forma o unitate echivalentă cu un nou enunț. Un astfel de grup este numit *enunț compus*. De exemplu, în limbajul LPS un enunț compus se obține dintr-o listă de enunțuri incluse între BEGIN și END :

$$<\textit{enunț compus}> : := \text{BEGIN} <\textit{listă de enunțuri}> \text{END}.$$

Enunțurile structurate și cele compuse sînt importante în scrierea de programe lizibile. Astfel, ele permit ca organizarea ierarhică naturală a celor mai mulți dintre algoritmi să fie direct reflectată în structura programului. În limbajele în care nu sînt definite astfel de enunțuri acest lucru se poate realiza numai prin ierarhii de subrutine, proceduri sau subprograme.

2.5.6. **Blocul.** Conceptul de bloc, ca element structural al programelor, va fi prezentat în continuare așa cum este el definit în limbaje ca ALGOL sau PL/I. Scopul lui este de a delimita domeniul de valabilitate al identificatorilor introduși pentru a face referiri la date, subrutine etc., așa încît aceștia să nu se aplice cu aceeași semnificație în întreg programul. Sintactic, blocul este delimitat prin parantezele **begin** și **end** în ALGOL, PROCEDURE ... END sau BEGIN ... END în PL/I etc. și începe cu declarații prin care se definesc identificatorii asociați lui.

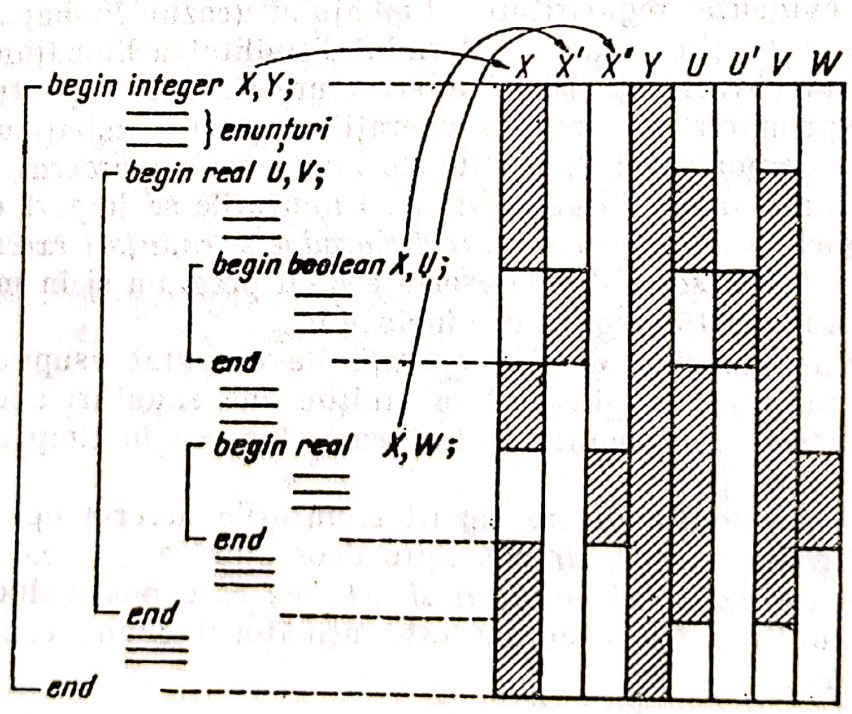



Fig. 2.1. Domeniul de valabilitate al variabilelor într-un program cu structură de blocuri.

numiţi şi *identificatori locali*. Aceasta înseamnă că unitatea reprezentată prin fiecare identificator declarat într-un bloc există numai în cadrul acestuia şi că orice entitate reprezentată printr-un acelaşi simbol în afara blocului este complet inaccesibilă în interiorul acestuia.

Figura 2.1 constituie o diagramă reprezentînd (prin porţiunile haşurate) domeniul de valabilitate al identificatorilor definiţi în diversele blocuri ale unui program scris în limbaj ALGOL. Blocurile sînt marcate lateral prin simbolul [.

În continuare se dau două din avantajele utilizării acestui element structural. 1) Modul în care este rezolvată problema valabilităţii identificatorilor permite scrierea programelor mari, compuse din blocuri incluse unele în altele, fără a exista pericolul ca definirea identificatorilor să fie contradictorie. 2) Faptul că un identificator este valabil numai în interiorul unui bloc, nu şi în afara lui, permite ca memoria necesară pentru variabilele dintr-un bloc să fie alocată cînd blocul este apelat, putînd fi eliberată la sfîrşitul apelării. Conceptul de bloc permite astfel utilizarea tehnicilor de alocare dinamică a memoriei în timpul execuţiei programelor. Cea mai folosită tehnică de alocare dinamică a memoriei are la bază conceptul de memorie *push-down*, noţiune strîns legată de structurile recursive.

2.5.7. **Procedura.** În general, o *procedură* constă dintr-o mulţime finită de instrucţiuni, fiecare dintre acestea putînd fi executată într-un interval de timp determinat. Procedura poate avea un număr finit de intrări şi ieşiri. Procedura (sau subrutină, subprogram) este un element structural important ce poate fi creat de către programator, instrucţiunile descriind operaţii realizabile pe un calculator virtual, definit de fiecare limbaj de programare. Pentru comunicarea cu procedura în cele mai multe cazuri se folosesc *parametri* sau *argumente*. Există şi cazuri de proceduri fără parametri.

În unele limbaje (de exemplu FORTRAN) procedura este o unitate structurală separată, în timp ce în altele (ALGOL, PL/I) poate fi inclusă în interiorul altor proceduri. Prin caracteristicile pe care la are procedura constituie un element de bază pentru programarea modulară, tehnică importantă pentru realizarea proiectelor de programare.

2.5.8. **Programul.** Un program este, în general, o entitate ce poate fi executată pe un calculator. Există multe încercări de a defini şi studia această noţiune. De exemplu, P. Wegner [92] asociază noţiunea de program cu cea de funcţie: „Un program într-un limbaj de programare poate fi privit ca o realizare a unei funcţii $f$ care, dată fiind o mulţime de date $x$, specifică o valoare $y=f(x)$; mulţimea tuturor datelor permise ca date de intrare formează *domeniul* programului, iar mulţimea ieşirilor



Scanned with OKEN Scanner

care pot fi obținute prin intermediul programului constituie *codomeniul* acestuia." Astfel, matematica pune la dispoziție o serie de noțiuni specifice ca: funcția, relația, algoritmul, schema logică, mașina abstractă, expresia, automatul etc., noțiuni ce permit reprezentarea programelor și a unor elemente asociate cum sînt enunțul, procedura etc. Studierea proprietăților programelor și a elementelor asociate se reduce astfel la studiul proprietăților noțiunilor matematice prin care acestea se reprezintă. De exemplu, folosind ca mijloc de reprezentare funcția sau, mai general, relația, ideea este următoarea: fiind dat un program $P$ se poate defini o funcție (relație) $F_P$ care descrie modul în care datele de intrare sînt transformate prin intermediul lui $P$, ori de cîte ori $P$ se termină. Rezultă deci că multe proprietăți ale lui $P$ pot fi exprimate în funcție de $F_P$. În consecință, $F_P$ poate reprezenta programul $P$, ori de cîte ori proprietățile exprimate sînt studiate.

Pentru ca acest mod de abordare să fie eficient este necesar să fie satisfăcută o condiție și anume: pentru orice program $P$ analizat, $F_P$ trebuie să fie dată într-o formă explicită și adecvată. Astfel, dacă $F_P$ se dă prin mulțimea perechilor ordonate $(x, y)$, așa încît plecînd cu $x$ după un interval de timp finit programul $P$ produce $y$, apoi oprindu-se, atunci această soluție nu va fi utilă în analiza lui $P$. Din contră, dacă $F_P$ este dată prin compunerea unor funcții primitive, folosind anumite operații standard, atunci într-adevăr $F_P$ poate reprezenta $P$ în multe cazuri. În acest fel analiza se face foarte ușor în cazul programelor structurate, putîndu-se aplica o procedură *bottom-up*, prin care se analizează fiecare element structural al acestora. Se deschide astfel calea spre demonstrarea corectitudinii programelor și se justifică o dată în plus utilizarea unor metode științifice de programare de genul *programării structurate*.

Observație. La multe sisteme de calcul în afara noțiunii de program se mai definește o altă unitate structurală de nivel superior, numită *lucrare*. O lucrare poate cuprinde unul sau mai multe programe ce urmează a fi executate secvențial sau în paralel. Limbajele utilizate pentru definirea lucrărilor sînt numite *limbaje de comandă* (job-control languages) care sînt, într-o anumită măsură, similare cu limbajele de programare.

2.5.9. **Segmentarea programelor.** În cele ce urmează se va analiza structura unui program în format executabil pe un calculator real. În cele mai multe cazuri programul se prezintă ca o singură unitate ce este încărcată în memorie și executată. Dacă zona-memorie disponibilă este insuficientă pentru a memora întreg programul, atunci este necesară divizarea acestuia în unități mai mici, numite *segmente*, care se încarcă pe rînd în memoria centrală. Acest lucru este posibil deoarece în momentul execuției, în general, nu este necesară prezența în memorie a întregului program.

Privit sub acest aspect, *segmentul este unitatea de încărcare în memoria centrală*, iar *programul este o mulțime cu o anumită structură ale cărei*



Scanned with OKEN Scanner

*elemente sînt segmente.* În acest caz se poate da următoarea definiţie pentru noţiunea de program.

*Programul* este o mulţime finită $P$ de unul sau mai multe segmente astfel încît:

— există un segment cu destinaţie specială, numit *segment rădăcină*, şi

— celelalte segmente sînt repartizate în $m \geq 0$ mulţimi disjuncte $P_1, P_2, \ldots, P_m$, fiecare dintre aceste mulţimi fiind la rîndul ei un program.

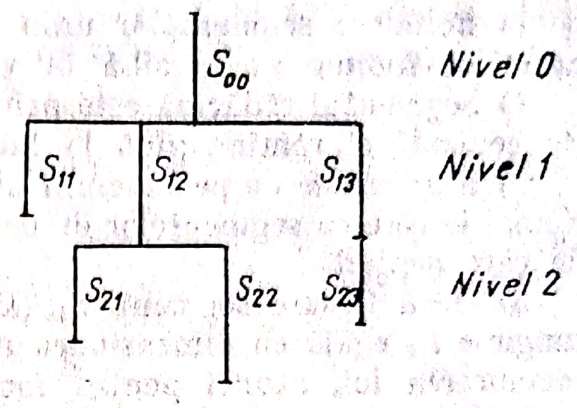



Fig. 2.2. Arborele de structură al unui program.

Această definiţie este recursivă; s-a definit un program funcţie de un program. O astfel de definiţie sugerează imediat că un program are o structură arborescentă, deci cea mai naturală modalitate pentru a-l reprezenta este un arbore care este numit *arbore de structură.*

Observaţie. Un punct de vedere similar privind noţiunea de program este prezentat de Th. Rus [65] care defineşte un program ca o mulţime de segmente, mulţime pe care sînt definite două operaţii: operaţia de concatenare a segmentelor şi operaţie de acoperire a segmentelor; definiţia conduce la acelaşi arbore de structură.

Un exemplu de reprezentare grafică pentru arborele de structură al unui program este cel din fig. 2.2, unde $S_{00}$ este segmentul rădăcină (uneori numit şi segment monitor sau supervizor); $S_{11}$, $S_{12}$, $S_{13}$ sînt segmente direct subordonate segmentului rădăcină şi aparţin nivelului 1 al structurii; $S_{21}$, $S_{22}$, $S_{23}$ aparţin nivelului 2 şi sînt subordonate direct segmentelor $S_{12}$ şi $S_{13}$. În sensul definiţiei $P=\{S_{00}, S_{11}, S_{12}, S_{13}, S_{21}, S_{22}, S_{23}\}$, iar $P_1=\{S_{11}\}$, $P_2=\{S_{12}, S_{21}, S_{22}\}$, $P_3=\{S_{13}, S_{23}\}$. $P_1$ privit ca un program conţine doar segmentul rădăcină $S_{11}$; în schimb $P_2$ conţine un segment rădăcină, $S_{12}$, şi două segmente subordonate, $S_{21}$, $S_{22}$, de acelaşi nivel, iar $P_3$ un segment rădăcină, $S_{13}$, şi un segment subordonat, $S_{23}$.

În unele cazuri arborele de structură al unui program se reprezintă liniar prin intermediul unei expresii ce conţine nume de segmente, paranteze rotunde şi virgule. De exemplu, arborele din fig. 2.2 se poate reprezenta prin expresia

$$S_{00}\,(S_{11},\ S_{12}\,(S_{21},\ S_{22}),\ S_{13}\,(S_{23})).$$

Regulile de alcătuire ale unor astfel de expresii rezultă din exemplul prezentat.

În definirea segmentelor unui program și a relațiilor dintre ele, în practică, trebuie să se aibă în vedere:

1) Segmentul rădăcină este primul segment care se încarcă în memoria centrală și rămîne pînă la încheierea execuției programului.

2) Segmentele de pe nivelul $i$ al arborelui de structură se pot încărca numai la cererea segmentelor de pe nivelul $i-1$ (nivelul imediat superior) de care depind.

3) Dacă fiecare segment $S_{ik}$ ($k=1, 2, \ldots, m$) de pe nivelul $i$ are o lungime $l_{ik}$ egală cu dimensiunea unei zone de memorie necesară pentru memorarea lui, atunci pentru încărcarea segmentelor $S_{ik}$ în memoria centrală se va rezerva o zonă unică de dimensiune

$$L_i = \max (l_{i1}, l_{i2}, \ldots, l_{im}).$$

4) Dacă arborele de structură al unui program $P$ are $p$ nivele, numerotate de la 0 la $p-1$, atunci pentru ca programul $P$ să poată fi executat este necesară o zonă-memorie de dimensiune $L$, unde

$$L \geqslant L_0 + L_1 + \ldots + L_{p-1}.$$

5. Din cele prezentate rezultă că la un moment dat în memorie nu pot exista decît segmente aparținînd unei singure *ramuri* a arborelui de structură; în cazul prezentat în fig. 2.2 ramurile sînt:

$$\{S_{00}, S_{11}\}, \{S_{00}, S_{12}, S_{21}\}, \{S_{00}, S_{12}, S_{22}\}, \{S_{00}, S_{13}, S_{23}\}.$$

Definirea segmentelor programelor se poate face de către programator sau automat. În primul caz, este necesară inserarea unor enunțuri specifice chiar la nivelul programului sursă. În al doilea caz, analiza programului pentru definirea segmentelor este automat realizată pe baza structurii logice a acestuia.

## 2.6. STRUCTURI DE DATE ÎN LIMBAJELE DE PROGRAMARE

Alegerea unui limbaj de programare implică stabilirea unei mulțimi de structuri de date specifice, iar înainte de a scrie programele este necesară adaptarea structurii datelor problemei de rezolvat la modalitățile de exprimare definite în limbajul de programare ales.

În cele ce urmează se vor face referiri la *datele elementare* și *structurile de date* în contextul limbajelor de programare. Accepțiunea în care aceste noțiuni sînt folosite depinde în mare măsură de fiecare limbaj în parte. Totuși un consens general există. De exemplu, sînt considerate drept



Scanned with OKEN Scanner

date elementare, în cele mai multe cazuri, numerele întregi, numerele reale, variabilele logice, etichetele, iar ca structuri de date tablourile, listele, articolele etc.

2.6.1. **Noţiunea de variabilă.** Noţiunea de variabilă are aproximativ acelaşi sens pe care îl are şi în matematică. Variabila permite scrierea de programe în care valorile curente ale unor date nu sînt cunoscute decît în momentul execuţiei. O *variabilă* se poate defini ca un triplet de forma

(nume, atribute, valori).

Atributele determină modul în care variabila este tratată în cadrul unui program. De exemplu, atributele determină mulţimea valorilor permise, modul de reprezentare a valorilor, valabilitatea numelui etc. Unul din cele mai importante atribute este *tipul variabilei*. Există limbaje care permit specificarea explicită doar a acestuia [96, 98], precizîndu-se astfel mulţimea valorilor pe care variabila le poate lua. Pe lîngă tip, o variabilă mai poate avea :

— atribute ce controlează domeniul de valabilitate al numelui variabilei ;

— atribute ce controlează crearea variabilei ;

— atribute pentru iniţializarea variabilelor (de exemplu, enunţurile DATA şi BLOCK DATA în limbaj FORTRAN, clauza VALUE în COBOL, atributul INITIAL în PL/I şi LPS etc.) ;

— atribute legate de structura memoriei, definită de unele limbaje de programare (de exemplu, enunţul EQUIVALENCE în FORTRAN, clauzele REDEFINES şi SYNCRONIZED în COBOL etc.).

În mod obişnuit, variabilele sînt specificate prin intermediul unor *enunţuri declarative* (*declaraţii*). O declarare de variabilă specifică numele şi o parte sau toate atributele asociate.

În ceea ce priveşte valorile unei variabile, acestea pot fi atribuite prin : *iniţializare* (cînd variabila este creată), *atribuire* şi *citire* de pe un suport extern, în timpul execuţiei programului. După atribuirea valorilor, o variabilă poate fi utilizată în alte enunţuri. O apariţie a numelui variabilei este folosită, în cele mai multe cazuri, pentru a referi valoarea (valorile) asociată (asociate).

2.6.2. **Noţiunea de tip.** Atributul *tip* defineşte : mulţimea valorilor pe care variabila le poate lua ; modul de reprezentare a valorilor ; cantitatea de memorie ce trebuie alocată ; modul în care operatorii aritmetici trebuie interpretaţi ; mulţimea operaţiilor la care variabila poate fi supusă.

Noţiunea de tip are o importanţă deosebită, aceasta fiind asociată nu numai unei variabile ci şi constantelor şi expresiilor. Tipul este declarat explicit în majoritatea limbajelor. În unele cazuri, de exemplu în



Scanned with OKEN Scanner

limbajul FORTRAN, tipul poate fi dedus și implicit (din prima literă a numelui unei variabile). Pentru limbajele ALGOL 60 și PASCAL asocierea tipului cu o variabilă se face printr-o declarație, iar în COBOL este dedus din clauzele PICTURE și USAGE asociate variabilelor.

O fundamentare riguroasă a conceptului de tip a fost dată de Hoare [30], Wirth [96, 98], Ledgard [37] și Scott [78].

2.6.3. **Noțiunea de constantă.** O constantă este o dată a cărei valoare nu poate fi schimbată. O clasă importantă de constante o formează *constantele literale*:

— *literale numerice*, de exemplu 150, —3.14159, 150E-6 etc.;
— *literale șir de caractere*, de exemplu: 'FELIX C-256', '30.10.76', 17HLITERAL HOLLERITH etc.

În limbajele de programare se întîlnesc și alte clase de constante:
— *constante logice*, de exemplu .TRUE. și .FALSE. în FORTRAN;
— *constante figurative*, de exemplu în limbaj COBOL: SPACE[S], ZERO[S], HIGH-VALUE[S], LOW-VALUE[S], QUOTE[S];
— *constante pointer*, de exemplu, NULL în limbajul PL/I.

În cazul în care constantele apar ca termeni în expresii este necesar să li se cunoască tipul, exact ca și în cazul variabilelor. Cele mai multe limbaje includ constante numerice de tip întreg sau real, distincția dintre ele fiind făcută prin caractere speciale (., E, D, +, — etc.).

2.6.4. **Tipuri de date elementare.** Drept date elementare sînt considerate de obicei următoarele tipuri:

a) *datele numerice* sau numerele, care au ca domenii de valori mulțimi finite de *numere întregi*, *numere reale* și *numere complexe*;

b) *datele logice*, care pot lua numai două valori: *adevărat* și *fals*;

c) *șirurile de caractere*, care sînt constituite din succesiuni finite de caractere;

d) *datele de control*, care există numai în contextul unui proces de calcul ([54], p. 161). Un exemplu îl constituie *etichetele*, care permit efectuarea de referiri la elemente ale programului. În cele mai multe dintre limbaje, o etichetă poate fi privită ca o simplă adresă. În cazul limbajelor în care figurează conceptul de bloc, etichetele reflectă contextul în care apar (adică în care au primit valorile inițiale). Astfel, dacă ieșirea dintr-un bloc ALGOL se face cu un enunț **go to**, efectul este același ca și în cazul în care ieșirea se asigură prin sfîrșitul normal al blocului.

2.6.5. **Noțiunea de structură de date.** În majoritatea aplicațiilor, în general, nu se lucrează direct cu date elementare, ci cu colecții de date



Scanned with OKEN Scanner

sau mulțimi ordonate de date care obișnuit sînt numite *structuri de date*. De obicei se consideră două tipuri de structuri de date:

a) structuri de date interne, create în memoria centrală (de exemplu tablourile, listele etc.);

b) structuri de date externe, create pe benzi sau discuri magnetice (de exemplu fișierele).

Pentru fiecare din aceste categorii se definesc tipuri de operații specifice. În general, cele două tipuri de structuri de date sînt tratate separat. Există și puncte de vedere de abordare unitară [96]; în acest caz fișierul este privit ca o structură de date ca și lista sau tabloul.

O caracteristică importantă a structurilor de date este posibilitatea de a le individualiza ca entități de sine stătătoare printr-un nume. Individualizarea elementelor componente ale unei structuri de date presupune însă și definirea tipului elementelor componente și a relației dintre acestea. Dacă toate elementele unei structuri sînt de același tip, se obține o *structură de date omogenă* (de exemplu tabloul); în caz contrar se obține o *structură de date eterogenă* (de exemplu articolul). Caracteristicile structurilor de date definesc operațiile cu elementele structurii de date și operațiile pe structuri de date.

Astfel, o structură de date poate fi privită ca o variabilă cu mai multe componente și anume ca o *variabilă structurată*. Pentru a defini o astfel de variabilă este necesar să se specifice tipul componentelor și metoda de structurare.

Metodele de creare și acces la structurile de date sînt în general mult mai complexe decît cele pentru datele elementare. Pentru a specifica acest lucru, McCarthy [13] descrie operațiile pe structuri de date prin funcții. Sînt definite următoarele tipuri de funcții:

– *funcții de creare*, care creează structuri de date;

– *funcții de suprimare*, care suprimă structuri de date;

– *funcții de acces*, care includ o largă varietate de metode de acces la structuri de date și anume: *acces prin nume*, *acces prin nume calificate*, *acces prin indici* etc. (Există două moduri în care o componentă a unei structuri de date poate fi adresată: 1) un mod ce determină accesul la valoarea componentei; 2) un mod ce determină accesul la adresa zonei de suport ce conține valoarea componentei; în cazul fișierelor cele două moduri de acces la articole obișnuit sînt numite: mod MOVE și mod LOCATE; fiind dat un *nume*, o funcție de acces îl transformă într-o *valoare* sau o *adresă*; de multe ori există și o funcție de acces inversă care, fiind dată valoarea unui element al structurii de date, determină adresa acestuia, mod de acces numit *acces asociativ*);

– *predicatele* – funcții folosite pentru a testa dacă un element al unei structuri are sau nu o anumită proprietate.



Scanned with OKEN Scanner

Tipurile de funcţii enumerate au fost definite de McCarthy în cadrul limbajului LISP, ele referindu-se direct la structurile de date numite *liste*.

Observaţie. Pe lîngă structurile de date unele limbaje de programare definesc şi *modele de memorie* specifice. Astfel, limbajul FORTRAN, în mod obişnuit, defineşte o memorie organizată pe cuvinte, fiecare cuvînt fiind identificabil printr-o adresă, cuvîntul constituind totodată şi unitatea de alocare, iar limbajul COBOL, o memorie organizată pe caractere, unitatea de alocare fiind o locaţie în care se poate memora un caracter. În aceste condiţii, în cazul unei structuri de date reale, pe lîngă relaţia dintre componente, implicată de structura însăşi, vor apărea relaţii suplimentare derivate din proprietăţile memoriei în care acestea sînt reprezentate. Aşa este cazul structurilor de date definite în limbajele COBOL, FORTRAN, PL/I, nu şi în limbaje ca ALGOL, APL, unde conceptul de memorie este complet evitat putîndu-se astfel defini structuri de date pure.

În continuare se vor prezenta pe scurt cele mai răspîndite structuri de date întîlnite în limbajele curente şi anume : şirul, tabloul, lista, articolul, fişierul.

2.6.6. **Şirul.** Un *şir* este o mulţime ordonată de elemente, care aparţin unei clase de obiecte elementare. Lungimea unui şir este egală cu numărul de elemente. Tipul unui şir este determinat de clasa de obiecte elementare căreia îi aparţin elementele şirului. Astfel, pot exista : şiruri de biţi, şiruri de numere, şiruri de litere, şiruri de numere şi litere, şiruri de caractere.

Cea mai importantă operaţie definită pe şiruri este operaţia de *concatenare* (vezi 2.3).

Nu toate limbajele de programare existente permit explicit prelucrări cu şiruri, de exemplu limbajele FORTRAN, ALGOL 60, COBOL. Limbajele PL/I şi LPS conţin facilităţi multiple în privinţa prelucrării de şiruri. Există şi limbaje specializate pentru astfel de prelucrări, cum este cazul limbajelor COMIT [99] şi mai ales SNOBOL [26]. S-a dovedit totuşi că şi un limbaj cum este FORTRAN-ul poate fi adaptat pentru operaţii cu şiruri [61].

2.6.7. **Tabloul.** Un *tablou* este o mulţime ordonată de elemente de acelaşi tip. Ordonarea elementelor într-un tablou este folosită pentru a individualiza elementele, în care scop, de obicei, se utilizează un ansamblu de indici (care în mod obişnuit sînt variabile întregi). Numărul de indici folosiţi pentru a identifica elementele unui tablou constituie *dimensiunea* tabloului. Astfel, există tablouri cu o dimensiune (*vectori*), tablouri cu două dimensiuni (*matrice*) etc. Datorită acestui mod specific de acces la elemente, tablourile mai sînt numite şi *variabile indexate*.

În definirea unui tablou trebuie să se specifice următoarele informaţii : numele tabloului, tipul elementelor componente, dimensiunea tabloului, numărul de elemente pentru fiecare dimensiune sau valorile maxime şi minime ale indicilor după fiecare dimensiune. Aceste informaţii sînt



Scanned with OKEN Scanner

necesare pentru a calcula memoria necesară memorării tabloului cît şi pentru generarea corectă a codului corespunzător operaţiilor definite pe tablouri.

Dat fiind faptul că tabloul este o structură de date care apare în majoritatea aplicaţiilor, aproape toate limbajele de programare permit definirea acestui tip de structură şi a unor operaţii pe tablouri.

2.6.8. **Lista.** O *listă* este o secvenţă finită de elemente, numite obişnuit componente. Aceste componente pot fi date elementare (întregi, caractere etc.) sau structuri de date (şiruri, tablouri, liste etc.). Nu este necesar ca toate componentele listei să fie de acelaşi tip. Listele au fost introduse din necesitatea realizării unor structuri de date dinamice (adică structura lor se poate modifica în cursul procesului de calcul), structuri necesare în special în domeniul inteligenţei artificiale. Ulterior, acest gen de structură a fost utilizat şi, de exemplu, la fişierele şi bazele de date.

Pentru a înţelege mai bine conceptul de listă cît şi diversele tipuri particulare de liste, se va da o definiţie mai exactă. Se consideră o mulţime finită $E=\{e_1, e_2, e_3, \ldots, e_n\}$ ale cărei elemente $e_i$ se numesc componente.

**Definiţie.** Se numeşte *listă liniară* o mulţime finită $E$ care satisface proprietăţile :

1) există o relaţie de ordine între elementele lui $E$, în aşa fel încît orice element $e_k$ are *predecesor* pe $e_{k-1}$ şi ca *succesor* pe $e_{k+1}$ ;

2) există un element în listă, şi anume $e_1$, care nu are nici un predecesor şi care este numit *capul listei* ;

3) există un element în listă, şi anume $e_n$, care nu are succesor şi care este numit *element terminal al listei* (de multe ori se notează cu NIL).

Pentru a indica succesorul sau predecesorul unui element al listei se foloseşte o informaţie adiţională numită *pointer* (care reprezintă o adresă). În cazul cel mai simplu elementele unei liste liniare pot fi perechi ordonate de forma $(d, p)$, unde $d$ este o dată elementară sau o structură de date, iar $p$ este un pointer ce indică succesorul elementului în listă (în cazul elementului terminal al listei $p$ are o valoare specială). O astfel de listă este numită *listă liniară simplă* sau *listă liniară asimetrică*, deoarece parcurgerea ei nu se poate face decît într-un singur sens, plecînd de la capul listei. Evident, timpul de parcurgere este proporţional cu numărul elementelor listei. Pentru reprezentarea grafică a unei astfel de liste se poate folosi conceptul de graf orientat, în care nodurile reprezintă date, iar arcele pointeri [81]. De exemplu, dacă se consideră o listă liniară simplă cu cinci elemente, aceasta se poate reprezenta grafic ca în fig. 2.3.

Dacă elementele unei liste liniare sînt formate din triplete de forma $(d, p_a, p_d)$, unde $d$ este o dată, $p_a$ este un pointer spre predecesor şi $p_d$ un



Scanned with OKEN Scanner

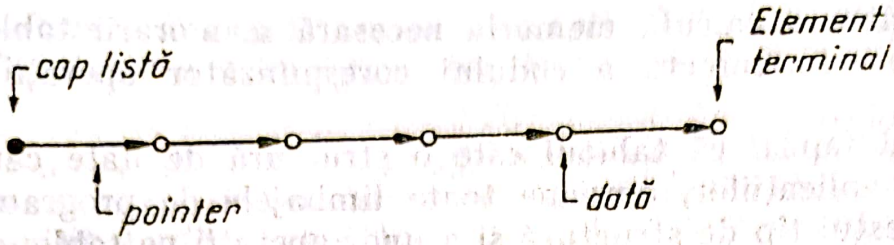



Fig. 2.3. Listă liniară simplă.

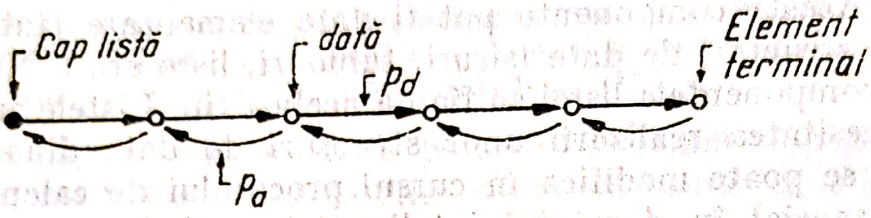



Fig. 2.4. Listă liniară simetrică.

pointer spre succesor, atunci se obţine o listă *liniară dublă înlănţuită* sau *listă liniară simetrică* care se poate reprezenta grafic la fel ca lista liniară simplă (fig. 2.4). O astfel de listă poate fi parcursă în ambele sensuri, deci prezintă două grade de libertate.

Dacă relaţia de ordine dintre elementele unei liste liniare este definită în aşa fel încît ultimul element are ca succesor primul element, atunci se obţine un nou tip de listă şi anume o *listă circulară* sau un *inel*. O listă circulară poate fi asimetrică sau simetrică, după cum elementele listei sînt dublete, sau respectiv, triplete, adică conţin un pointer sau doi pointeri. Figura 2.5 prezintă grafurile asociate la două liste de acest tip.

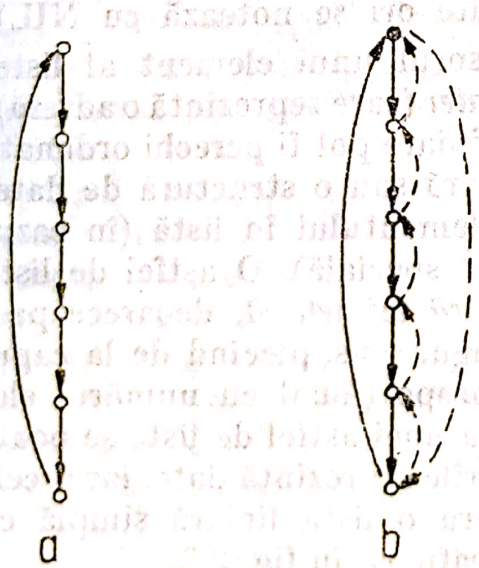



Fig. 2.5. Liste circulare :
a — listă circulară asimetrică;
b — listă circulară simetrică.

*Reprezentarea pe suport a listelor* se poate face în două moduri, care sînt prezentate în continuare.

1) *Reprezentarea liniară*. În acest caz elementele listei se reprezintă în zone-memorie consecutive. Pointerii nu se reprezintă, iar ordinea logică dintre elemente este identică cu ordinea fizică. Dacă toate elementele se reprezintă în zone-memorie de lungime $L$, atunci :

$$\text{adresa } (e_1) = \text{adresa capului listei şi}$$
$$\text{adresa } (e_k) = \text{adresa } (e_{k-1}) + L, \quad \text{pentru } k \geq 2,$$

unde prin *adresa* $(e_i)$ s-a notat adresa zonei-memorie destinată elementului $e_i$ din listă.

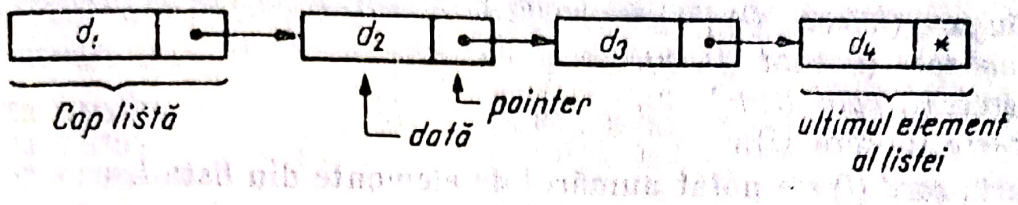



Fig. 2.6. Listă liniară.

2) *Reprezentarea înlănțuită.* În acest caz elementele listei se reprezintă în diverse zone-memorie. Pointerii se reprezintă efectiv, ei fiind adresele zonelor-memorie unde se află succesorii sau predecesorii elementelor. În acest fel se asigură o independență a ordinii logice a elementelor în listă în raport cu ordinea pe suport. O listă liniară se mai poate reprezenta ca în fig. 2.6.

Cîteva din *operațiile cele mai importante definite pe liste liniare* sînt:

*creare* ($l$) — creează o listă $l$;
*suprimare* ($l$) — suprimă o lista $l$:
*valoare* ($l$, $r$) — furnizează valoarea elementului de rang $r$ din lista $l$;
*card* ($l$) — dă numărul de elemente din lista $l$;
*adăugare* ($l$, $r$) — adaugă la lista $l$ un nou element de rang $r$ și valoare nedefinită;
*actualizare* ($l$, $r$, $v$) — înlocuiește valoarea actuală a elementului de rang $r$ din lista $l$ cu valoarea $v$;
*ștergere* ($l$, $r$) — suprimă din lista $l$ elementul de rang $r$;
*salvare* ($l_1$, $l_2$) — copiază lista $l_2$ în lista $l_1$.

Se remarcă faptul că inserarea unui nou element într-o listă, adică operația:

*inserare* ($l$, $r$, $v$)

se poate realiza prin succesiunea de operații:

*adăugare* ($l$, $r$),
*actualizare* ($l$, $r$, $v$).

În cazul în care se impun anumite restricții asupra operațiilor de inserare, ștergere sau consultare a elementelor unei liste liniare, se obțin cîteva tipuri particulare de liste folosite frecvent în practică.

a) *Lista* LIFO (Last-In, First-Ont) este o listă liniară în care inserările, ștergerile și regăsirea elementelor se fac la un singur capăt al listei. În literatura de specialitate, lista LIFO se mai întîlnește și sub denumirea de *pilă, stivă* sau *stack*. Operațiile definite pentru listele liniare se particularizează, pentru listele LIFO, în felul următor:

*creare* ($l$),
*suprimare* ($l$),
*valoare* ($l$, *card* ($l$)),

*adăugare* (*l*, *card* (*l*)+1),
*actualizare* (*l*, *card* (*l*), *v*),
*inserare* (*l*, *card* (*l*)+1, *v*),
*ștergere* (*l*, *card* (*l*)),

unde prin *card* (*l*) s-a notat numărul de elemente din lista *l*.

O astfel de listă se utilizează frecvent pentru scrierea de *programe reentrante* (programe ce nu sînt modificate prin propria execuție) și în cadrul procesului de compilare.

b) *Lista* FIFO (First-In, First-Out), este o listă liniară în care inserările de elemente se fac la un capăt al listei (sfîrșitul listei), iar regăsirile și ștergerile de elemente se fac la celălalt capăt al listei (capul listei). Deoarece se regăsește întotdeauna primul element adăugat, o astfel de listă se mai numește și *coadă* sau *șir de așteptare*. Operațiile specifice pentru o astfel de listă sînt următoarele :

*creare* (*l*),
*suprimare* (*l*),
*adăugare* (*l*, *card* (*l*)+1),
*actualizare* (*l*, *card* (*l*), *v*),
*inserare* (*l*, *card* (*l*)+1, *v*),
*ștergere* (*l*, 1),
*valoare* (*l*, 1).

O astfel de listă este frecvent utilizată în sistemele de operare (în rutinele de alocare a resurselor) sau în programe de simulare.

c) *Lista cu două capete* este o listă liniară în care inserările, ștergerile și regăsirile de elemente se fac la ambele capete ale listei.

2.6.9. **Articolul.** Un *articol* (sau o structură cum este numit în PL/I sau în LPS) poate fi privit ca o colecție de date la care se adaugă mulțimea numelor și a relațiilor folosite pentru acces la componente [54]. Articolul este, în general, o structură de date eterogenă, adică elementele sale nu sînt neapărat de același tip. Elementele unui articol pot fi date elementare sau structuri de date (de exemplu tablouri).

În COBOL sau PL/I aceste structuri de date sînt strîns legate de *modelul de memorie* definit de fiecare limbaj în parte. Din această cauză accesul la elementele unui articol se obține prin adrese relative, iar articolul însuși poate fi utilizat ca bază pentru operațiile de intrare/ieșire (citiri sau scrieri de articole).

Prin structura sa, articolul este un exemplu de structură recursivă ca și expresiile sau blocurile. Într-un articol elementele sînt ordonate pe nivele, fiecare nivel fiind caracterizat printr-un număr, numit *număr de nivel*. Acest număr de nivel este asociat tuturor elementelor ce aparțin aceluiași nivel.



Scanned with OKEN Scanner

Un exemplu de articol exprimat în limbajul COBOL este următorul:

```
01 PERSOANA.
   02 NUME                    PIC X (15).
   02 VÎRSTA                  PIC 9 (3).
   02 ADRESA.
      03 JUDET                PIC A (15).
      03 LOCALITATE           PIC X (15).
      03 NUME-STRADA          PIC X (20).
      03 NR-STRADA            PIC 999.
   02 DATA-NAȘTERE.
      03 AN                   PIC 99.
      03 LUNA                 PIC 99.
      03 ZI                   PIC 99.
   02 DATA-ANGAJARE.
      03 AN                   PIC 99.
      03 LUNA                 PIC 99.
      03 ZI                   PIC 99.
```

Se remarcă faptul că numai variabilelor ce aparțin ultimelor nivele li se declară tipul.

Accesul la elementele unui articol se realizează prin intermediul numelor de variabile, iar în cazul în care există mai multe variabile cu același nume, prin *nume calificate*, de exemplu:

```
AN OF DATA-NAȘTERE
```

sau

```
AN IN DATA-ANGAJARE.
```

2.6.10. **Fișierul.** Un *fișier* este o colecție organizată de date elementare. Organizarea fișierului determină modul de acces la elementele acestuia. Pentru a da o definiție mai exactă a noțiunii de fișier se consideră o mulțime finită $O$ de obiecte (sau entități), care trebuie descrise din punct de vedere informațional:

$$O=\{o_i \mid i=1,\ 2,\ \ldots,\ n \text{ și } n\geq 1\},$$

unde cu $o_i$ s-au notat obiectele mulțimii $O$. Elementele mulțimii $O$ au comun faptul că într-un anumit caz acestea pot fi caracterizate printr-o mulțime finită de proprietăți:

$$P=\{p_1,\ p_2,\ \ldots,\ p_m \mid m\geq 1\}.$$

De exemplu, dacă $O$ este mulțimea persoanelor dintr-o unitate economică, atunci $P$ poate fi

$$P=\{\textit{nume, dată-naștere, adresă, profesie, sex, vîrstă}\}.$$



Scanned with OKEN Scanner

Aceste proprietăţi pot fi cantitative (de exemplu: *greutate, înălţime, vîrstă*) sau calitative (*sex, profesie*). Dacă se notează cu $v_{ik}$ valoarea asociată proprietăţii $p_k$ pentru obiectul $o_i$, atunci un fişier se poate defini ca o mulţime $F$ de triplete de forma $(o_i, p_k, v_{ik})$. Deci

$$F=\{(o_i,\ p_k,\ v_{,ik}) \mid i=1,\ 2,\ \ldots,\ n,\ \ k=1,\ 2,\ \ldots,\ m\}.$$

Din această cauză uneori un fişier este numit şi *data set*, un triplet fiind de fapt o dată exprimată sub forma (*obiect, atribut, valoare*).

Fie $V_k$ mulţimea valorilor pe care proprietatea $p_k$ le poate lua, adică

$$V_k=\{v_{ik} \mid i \in \{1,\ 2,\ \ldots,\ n_i\} \text{ şi } n_i \leqslant n\} \cup \{\wedge\},$$

unde cu $\wedge$ s-a notat o valoare specială care este asociată unei proprietăţi $p_k$ atunci cînd nu se cunoaşte pe moment valoarea acesteia pentru un obiect $o_i$. Se consideră de asemenea o aplicaţie injectivă $f$ a mulţimii $O$ pe mulţimea $V_1 \times V_2 \times \ldots \times V_m$ (produsul cartezian al celor $m$ mulţimi); atunci $f(O)$ este imaginea mulţimii $O$ prin $f$ şi

$$f(O) \subseteq V_1 \times V_2 \times \ldots \times V_m.$$

Evident, fiecărui element din $O$ i se asociază prin $f$ un element din $V_1 \times \times V_2 \times \ldots \times V_m$. Un element $f(o_i) \in f(O)$, obţinut ca o succesiune de elemente din $V_1, V_2, \ldots, V_m$, este denumit *înregistrare logică*. O înregistrare logică este deci asociată unui element din $O$ (adică unui obiect) şi constituie o caracterizare informaţională a acestuia. În cele mai multe cazuri de fişiere singurele elemente care sînt explicit reprezentate sînt înregistrările logice, astfel de fişiere fiind numite *fişiere omogene* [8]. În acest caz este foarte important ca valorile să fie ordonate astfel încît corelarea cu proprietăţile să se poată face atunci cînd este necesar. De obicei înregistrările logice se divizează în *cîmpuri*, fiecare cîmp reprezentînd o valoare asociată unei proprietăţi.

Totuşi mai corectă este definirea în cadrul fişierelor a noţiunii de articol, un articol $r_i$ fiind un triplet de forma

$$r_i=(o_i,\ P,\ f(o_i))$$

Utilizarea acestei noţiuni este necesară deoarece permite o mai bună individualizare a înregistrărilor logice ale fişierului, cît şi a datelor cuprinse în fiecare articol.

**Exemplu.** Fie $O=\{o_1, o_2, o_3, o_4, o_5\}$ o mulţime de cinci persoane:
$P=\{nume, direcţie, profesie, retribuţie\}$
şi:
$V_1=\{$DUMITRU, POPESCU, IONESCU$\}$,
$V_2=\{$TEHNICA, CERCETARE, PROIECTARE, CONDUCERE$\}$,
$V_3=\{$MAT, EC, FIZ, ING$\}$,
$V_4=\{1\,750,\ 2\,430,\ 3\,000,\ 3\,500\}$.



Scanned with OKEN Scanner

Fișierul $F$ creat, plecînd de la aceste elemente, va avea cinci articole după cum urmează :

| | *nume* | *direcție* | *profesie* | *retribuție* |
|---|---|---|---|---|
| $f(o_1)$ : | DUMITRU | TEHNICA | MAT | 2 430 |
| $f(o_2)$ : | DUMITRU | CERCETARE | ING | 3 000 |
| $f(o_3)$ : | IONESCU | CONDUCERE | EC | 2 430 |
| $f(o_4)$ : | IONESCU | PROIECTARE | ING | 1 750 |
| $f(o_5)$ : | POPESCU | TEHNICA | FIZ | 3 500 |

Astfel, articolul $r_1$ este

$$r_1=(o_1, \underbrace{\{nume, direcție, profesie, retribuție\}}_{P}, \underbrace{(DUMITRU, TEHNICA, MAT, 2\,430)}_{f(o_1)})$$

În general, înregistrările logice ale unui fișier pot fi de *lungime fixă* sau de *lungime variabilă*. Un fișier omogen constă din înregistrări logice de lungime fixă. Pe suport înregistrările logice pot fi grupate în unități distincte numite *blocuri* sau *înregistrări fizice*. Înregistrarea fizică constituie unitatea de informație transferată în cursul unei operații de intrare/ieșire.

Numărul de înregistrări logice cuprinse într-o înregistrare fizică este numit *factor de blocare*, iar zonele din memoria centrală în care sînt pregătite înregistrările fizice ce urmează a fi transmise spre un periferic, sau în care se memorează înregistrările fizice citite de pe un suport extern sînt numite *zone-tampon*. La cele mai multe dintre limbaje, utilizatorii nu au acces direct la înregistrările fizice ci la articole (sau elemente ale acestora), în zone-memorie distincte numite *zone-articol*.

Astfel, dacă se face studiul organizării unui fișier (organizarea determinînd metoda sau metodele de acces la articolele fișierului), atunci studiul trebuie făcut la două nivele și anume :

– un *nivel logic* corespunzător organizării logice a fișierului ;

– un *nivel fizic* corespunzător organizării pe suport a fișierului.

2.6.10.1. *Organizarea fișierelor*. La nivel logic un fișier este privit ca o mulțime ordonată de articole, unei organizări a fișierului fiindu-i caracteristică una sau mai multe metode de acces la articole. La rîndul ei, organizarea unui fișier este determinată de relația dintre articole. În majoritatea limbajelor de programare, la nivel logic se consideră trei tipuri de organizare a fișierelor și anume : organizarea secvențială, organizarea selectivă sau directă, organizarea indexată.

a) *Organizarea secvențială*. În general, se spune că un *fișier* $F$ este *organizat secvențial*, dacă :

1) între articolele sale este definită o relație de ordine $<$ astfel încît

$$r_1<r_2\ldots<r_n,$$

unde cu $r_i$ $(i=1, 2, \ldots, n)$ s-au notat articolele fișierului $F$ ;

2) articolele sînt memorate pe suport, în ordinea definită, în zone succesive sau în zone înlănțuite.

Deși există excepții (de exemplu, ordonarea articolelor fiind determinată de momentele în care ele sînt incluse în fișier), de obicei, într-un fișier secvențial articolele sînt ordonate după valorile unei date care apare în toate articolele și care obișnuit este numită *cheie*. De exemplu, în cazul unui fișier de personal (fiecare articol corespunzînd unei persoane) se poate face o ordonare alfabetică a articolelor după valorile cheii *nume*. În cazul în care o cheie are aceeași valoare pentru mai multe articole s-ar putea ca ordonarea după valorile ei să nu fie satisfăcătoare. În acest caz se poate utiliza o a doua cheie de ordonare a articolelor pentru care prima cheie are aceeași valoare.

Operația prin care articolele unui fișier sînt ordonate după una sau mai multe chei este numită *sortare*.

În cazul organizării secvențiale a fișierelor singurul mod de acces la articolele fișierului este *accesul secvențial* care presupune accesul la articolul $r_i$ după consultarea articolelor $r_1, r_2, \ldots, r_{i-1}$. Astfel, dacă fiecare din cele $n$ articole ale fișierului este solicitat cu aceeași probabilitate, atunci căutarea unui articol necesită în medie un timp proporțional cu $(n-1)/2$. Pentru a minimiza timpul de acces în anumite cazuri (de exemplu fișiere pe suport adresabil) se poate defini un mod de acces mai rapid bazat pe metoda *căutării logaritmice* [33] (sau *căutare binară*), care conduce la un timp de căutare proporțional cu $[\log_2 n]+1$, unde $[\log_2 n]$ reprezintă partea întreagă din $\log_2 n$.

b) *Organizarea selectivă*. Fie o mulțime $R=\{r_i \mid i=1, 2, \ldots, n\}$ de articole, fiecare articol fiind identificat printr-o valoare unică $k_i$ asociată unei chei ; mulțimea valorilor acestei chei se va nota cu $K$. Fie de asemenea o funcție surjectivă $f: K \to I$, unde cu $I$ s-a notat mulțimea numerelor întregi din intervalul $[0, m-1]$, iar $m$ este un număr întreg.

Un dublet $F=(R, f)$ este un fișier de organizare selectivă, dacă :

1) în mulțimea $R$ este definită o partiție $\{R_0, R_1, \ldots, R_{m-1}\}$ ;

2) prin funcția $f$ fiecărui articol i se asociază submulțimea $R_i$ din care face parte ;

3) între articolele fiecărei submulțimi $R_i$ este definită o relație de ordine.

Articolele care aparțin unei submulțimi $R_i$ sînt numite *articole sinonime*, iar funcția $f$ astfel definită este numită *funcție hash* sau *funcție de randomizare*. Numărul $m$ de submulțimi ale partiției se alege astfel încît $m \leqslant n$, pentru a evita ca submulțimile $R_i$ să fie vide.

În cazul fișierelor selective singurul mod de acces la articole este *accesul direct*. Accesul la un articol $r_i$ se obține astfel : fiind dată valoarea $k_i \in K$, ce identifică articolul $r_i$, atunci $f(k_i)$ este un număr $j \in I$ ce repre-

zintă submulțimea $R_j$ din care face parte $r_i$; între articolele din $R_i$ existînd o ordonare, sînt suficiente în medie $(|R_j|-1)/2$ comparări pentru a găsi pe $r_i$ (cu $|R_j|$ s-a notat cardinalul mulțimii $R_j$).

Pentru memorarea articolelor unei submulțimi $R_i$ se rezervă o zonă de suport de dimensiune fixă care obișnuit este numită *căsuță*. Astfel, mulțimea $I$ identifică $m$ zone de suport succesive, de aceeași dimensiune, asociate respectiv submulțimilor $R_0, R_1, \ldots, R_{m-1}$. Cele $m$ zone de suport constituie *zona principală* a fișierului.

Articolele sinonime ce aparțin unei submulțimi $R_i$ sînt memorate secvențial în căsuța $i$. În cazul în care un articol sinonim nu mai are loc în căsuță, atunci apare *o depășire de capacitate* sau o *coliziune*. O astfel de situație se rezolvă, de obicei, în două moduri: se memorează articolul în una din celelalte căsuțe care nu este completată cu articole sau se memorează articolul într-o zonă de suport separată, numită *zonă de depășire* [79].

În primul caz articolul este memorat în „următoarea" căsuță incompletă. Aici prin „următoarea" se înțelege una din căsuțele cu un număr asociat $i$ mai mare decît cel asociat căsuței unde s-a produs depășirea; pentru ultima căsuță (căsuța $m-1$), „următoarea" este căsuța 0. În al doilea caz articolul memorat în zona de depășire, asigurîndu-se legătura cu căsuța prin pointeri.

Se remarcă faptul că prin coliziune, indiferent de modul cum aceasta este rezolvată, se complică accesul la articole, mărind totodată și timpul de acces. Din această cauză căsuțele trebuie în așa fel dimensionate, pentru o funcție $f$ dată, încît să evite pe cît posibil apariția de coliziuni, chiar cu riscul unui consum mai mare de suport.

c) *Organizarea indexală.* Se consideră o mulțime de articole $R=\{r_i \mid i=1, 2, \ldots, n\}$, fiecare articol $r_i$ fiind identificat printr-o valoare unică $k_i$ asociată unei chei. Fie de asemenea o mulțime $I=\{(k_i, p_i) \mid i=1, 2, \ldots, m\}$, unde $p_i$ este un pointer spre articolul $r_i$. O pereche $(k_i, p_i)$ este numită *index*. Mulțimea $I$ este ordonată după valorile primei componente a fiecărei perechi.

Se numește *fișier indexat* o pereche $(R, I)$, $I$ constituind mulțimea de indecși asociată fișierului.

O astfel de organizare determină o metodă simplă de *acces direct* la articole și anume: fiind dată o valoare $k_i$, se caută mai întîi în mulțimea $I$ perechea $(k_i, p_i)$ care are prima componentă identică cu valoarea dată; pentru aceasta se poate folosi o metodă de căutare rapidă de tipul căutării logaritmice (mulțimea $I$ este ordonată). Odată perechea $(k_i, p_i)$ localizată, accesul direct la articol se obține folosind cea de-a doua componentă și anume pointerul $p_i$.



Scanned with OKEN Scanner

În cazul în care este posibil și accesul secvențial la articole (deci fără a se utiliza mulțimea de indecși), se spune că fișierul este de *organizare secvențială – indexată*, modul de acces la articole fiind numit *acces mixt*.

Mulțimea $I$ este memorată pe suport sub forma unui tabel, numit *tabel de indecși*, a cărui dimensiune este direct proporțională cu numărul de articole din $R$.

O variantă perfecționată a acestui mod de organizare se obține în felul următor:

1) pe mulțimea $R$ se definește mai întîi o relație de ordine $<$ astfel încît $r_1 < r_2 < \ldots < r_n$; ordonarea se face după valorile $k_i$ ale unei chei;

2) în mulțimea $R$ se definește apoi o partiție $\{R_1, R_2, \ldots, R_m\}$ astfel încît $r_i < r_j$ oricare ar fi $r_i \in R_k$ și $r_j \in R_{k+1}$ $(k=1, 2, \ldots, m-1)$;

3) se definește o mulțime de indecși $I=\{(k^i, p_i) \mid i=1, 2, \ldots, m\}$, fiecare index fiind asociat unei submulțimi $R_i$ și constituindu-se din valoarea cea mai mare a cheii, pe care o au articolele din $R_i$ și adresa pe suport, unde se află elementele ce aparțin lui $R_i$ (adică un pointer). În aceste condiții dimensiunea tabelului de indecși se reduce mult și mulțimea $I$ devine independentă de numărul de articole din $R$.

Procedura de căutare în acces direct a unui articol devine atunci următoarea: se compară secvențial valoarea dată $k_i$ cu prima componentă a fiecărui index pînă cînd relația $<$ nu mai este adevărată; fie $(k^j, p_j)$ acest index, atunci articolul $r_i \in R_{j-1}$, deci pointerul $p_{j-1}$ din indexul $(k^{j-1}, p_{j-1})$ dă adresa pe suport a submulțimii $R_{j-1}$, astfel că prin încă cel mult $|R_{j-1}|$ comparări articolul căutat este găsit.

În unele cazuri pentru a reduce și mai mult timpul de căutare în tabelul de indecși se realizează o indexare după procedeul descris chiar a mulțimii $I$, obținîndu-se astfel un tabel de indecși cu două nivele.

**Observație.** În cazul în care un articol este adăugat, acesta se memorează într-o zonă de depășire, fiind înlănțuit prin adrese în așa fel încît relația de ordine definită pe $R$ să se păstreze.

2.6.10.2. *Operații pe fișiere.* a) *Crearea* unui fișier este operația prin care fișierul, în stare inițială, este memorat pe un suport. În cazul fișierelor de organizare secvențială, de exemplu, articolele se memorează în zone succesive de suport sau în zone de suport înlănțuite, conform ordinii definite.

b) *Consultarea* unui fișier este operația prin care articolele unui fișier sînt selectate (conform modului de acces caracteristic organizării sale) în vederea efectuării unui proces de prelucrare definit asupra întregului fișier sau numai asupra unei părți a acestuia.

c) *Actualizarea* unui fișier este operația prin care: se pot adăuga noi articole, se pot anula acele articole care nu mai conțin date actuale, se pot modifica, în unele articole, date care nu mai sînt actuale. Această



Scanned with OKEN Scanner

operaţie permite evoluţia unui fişier, prin intermediul ei obţinîndu-se o nouă stare a acestuia.

d) *Copierea* unui fişier este definită ca fiind operaţia prin care se obţine o copie identică a acestuia pe un alt suport.

e) *Salvarea* unui fişier este operaţia prin care acesta este copiat pe un alt suport (de obicei bandă magnetică); copia astfel obţinută nu poate fi utilizată direct în procesul de prelucrare.

f) *Restaurarea* unui fişier este operaţia inversă salvării, prin care un fişier, salvat anterior pe un suport, este pregătit în vederea utilizării.

g) *Sortarea* unui fişier este operaţia de ordonare a articolelor unui fişier după valorile uneia sau a mai multor chei.

h) *Ventilarea* este operaţia prin care plecînd de la un fişier $F$ se obţin mai multe fişiere.

i) *Regruparea sau reuniunea* este operaţia prin care plecînd de la mai multe fişiere se obţine un singur fişier. Cînd fişierele iniţiale şi fişierul rezultant sînt sortate, operaţia este numită *fuzionare* sau *interclasare*.

2.6.10.3. *Memorarea fişierelor pe suport.* Pentru identificarea fişierelor pe suporturi acestea sînt însoţite de o informaţie specială, numită *informaţie de identificare*. Aceasta este memorată sub forma unor înregistrări fizice speciale, numite *etichete*. Etichetele pot avea o structură standard sau nu. *Etichetele standard* sînt alcătuite de către sistemele de gestiune ale fişierelor care, ulterior, le utilizează pentru identificarea şi validarea accesului la fişiere. *Etichetele nestandard* au o structură definită de utilizator, din care cauză mai sînt numite şi *etichete utilizator*.

Pe suport fişierele pot ocupa unu sau mai multe volume, prin *volum* înţelegînd o unitate de suport (rolă de bandă magnetică, pilă de discuri magnetice etc.). Dacă pe un volum se află doar un fişier, se spune că există o structură de tip *monofişier-monovolum*; în practică se mai pot alcătui structuri de tip *monofişier-multivolum* (un fişier pe mai multe volume), *multifişier-monovolum* (mai multe fişiere pe un singur volum) şi *multifişier-multivolum* (mai multe fişiere pe mai multe volume).

La memorarea pe suport a articolelor unui fişier trebuie să se aibă în vedere, în principal, următoarele obiective:

1) minimizarea necesarului de suport;
2) reducerea numărului de operaţii de intrare/ieşire;
3) minimizarea timpului de acces la înregistrări.

## 2.7. STRUCTURI DE CONTROL ÎN LIMBAJELE DE PROGRAMARE

În continuare se va discuta ordinea în care operaţiile de bază ale unui program, descrise prin enunţuri, sînt executate, adică modul în care se transferă controlul de la un enunţ la altul, sub aspectul execuţiei



Scanned with OKEN Scanner

programului. Structura implicată prin această ordonare a execuției enunțurilor este numită *structura de control* a programului. În limbajele de programare structura de control constituie mijlocul prin care se realizează ramificări (salturi) condiționate și necondiționate, cicluri cu număr fix sau variabil de pași și creează de asemenea posibilitatea de a putea reutiliza părți ale programului.

În cazul cel mai simplu de structură de control operațiile descrise de enunțuri (enunțuri executive) sînt executate secvențial, în ordinea în care enunțurile apar în program. Enunțurile unui program cu o astfel de structură de control se pot ordona într-o listă liniară simplă $\{e_1, e_2, \ldots, e_n\}$ astfel încît: $e_1$ este primul enunț care se execută; execuția enunțului $e_i$ antrenează execuția enunțului $e_{i+1}$; $e_n$ este ultimul enunț care se execută. Acest model nu este adecvat pentru aplicațiile practice deoarece cere ca fiecare operație să fie inclusă explicit în program. Este necesară deci introducerea unor modele mai complicate în care determinarea succesorului unui enunț să nu fie legată întotdeauna de ordinea în care apar enunțurile în program. În acest caz se impune separarea *ordinii sintactice* a enunțurilor de ordinea *executării* lor. Pentru aceasta este necesară includerea în program a unor enunțuri care să permită determinarea (condiționat sau nu) a succesorului unui enunț în momentul execuției, sau care să permită executarea, de mai multe ori, a unui enunț sau a unui grup de enunțuri. Aceste enunțuri vor fi numite *enunțuri de control*.

Astfel dacă, din punctul de vedere al ordinii sintactice, enunțurile unui program formează o listă liniară simplă, pentru a modela ordinea execuției enunțurilor se folosește un *graf orientat* (vezi fig. 2.7), un drum în acest graf fiind o posibilă execuție a programului.

Fie $E$ mulțimea enunțurilor unui program, care, sintactic, formează o listă liniară simplă

$$E = \{e_1, e_2, e_3, \ldots, e_n\}.$$

Fie de asemenea $\Gamma$ o corespondență a lui $E$ în $E$ care pune în relație fiecare $e_i$ cu un anumit număr de elemente din $E$ (eventual cu nici unul).

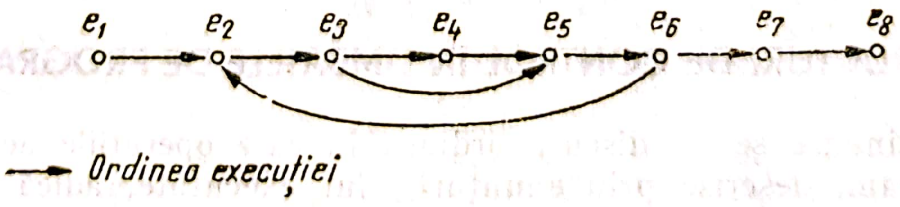



Fig. 2.7. Ordinea executării enunțurilor într-un program.



Scanned with OKEN Scanner

Un element $e_j \in \Gamma(e_i)$, dacă, în cursul execuției programului, enunțul $e_j$ poate succede imediat enunțului $e_i$. Astfel:

1) în general $\Gamma(e_i)=\{e_{i+1}\}$;

2) în cazul unui salt necondiționat $\Gamma(e_i)=\{e_j\}$ cu $j\neq i$, dacă $j<i$, este vorba de un ciclu;

3) în cazul unui salt condiționat sau test avem, de exemplu, $\Gamma(e_i)=\{e_j, e_k\}$;

4) în cazul unui enunț final $\Gamma(e_i)=\varnothing$.

Aproape toate limbajele de programare conțin următoarele tipuri simple de enunțuri de control al ordinii execuției:

a) enunțul GO TO pentru salt necondiționat;

b) enunțul IF pentru transfer condiționat;

c) enunțuri de ciclare (de tip DO) pentru cicluri controlate;

d) enunțuri pentru apel de proceduri, subprograme.

Un alt tip de structură control este obținut prin monitorizarea execuției programelor și efectuarea de salturi necondiționate spre secvențe de enunțuri, în cazul în care sînt îndeplinite anumite condiții (de exemplu apariția unei erori în cursul unei operații de intrare/ieșire sau chiar în cursul execuției programului, detectarea sfîrșitului unui fișier secvențial etc.).

De asemenea, un alt tip de structură de control este cel utilizat pentru controlul proceselor paralele [27], adică execuții de programe care se desfășoară simultan în timp și nu secvențial.

Pentru reprezentarea grafică a ultimelor două tipuri de structuri de control se utilizează de obicei așa-numitele *rețele Petri* [44], care permit o explicitare mai bună a monitorizării și controlului execuției proceselor paralele.

2.7.1. **Enunțul** GO TO este cel mai simplu tip de enunț de control. Enunțul GO TO apare aproape în toate limbajele de programare și are forma generală:

GO TO < *elichelă-enunț* >

Dacă $e_i$ este enunțul GO TO și $e_j$ enunțul referit prin GO TO, atunci $\Gamma(e_i)=\{e_j\}$ cu $i\neq j$.

Unele limbaje de programare conțin și o variantă extinsă a enunțului GO TO în care saltul are loc la un enunț dintr-o listă precizată la început. Fie $(e_{j1}, e_{j2}, \ldots, e_{jn})$ această listă și $j$ o variabilă întreagă ce poate lua valori în mulțimea $\{1, 2, \ldots, n\}$. În acest caz

$$\Gamma(e_i)=\{e_{j1}, e_{j2}, \ldots, e_{jn}\}.$$



Scanned with OKEN Scanner

Dacă variabila $j$ ia valoarea $k$, atunci saltul are loc la enunțul $e_{jk}$; în cazul în care $j \notin \{1, 2, \ldots, n\}$, atunci se execută, de obicei, enunțul $e_{i+1}$. De exemplu, în cazul enunțului

GO TO (100, 110, 90, 120), J

din limbaj FORTRAN, dacă $J=2$, se va executa enunțul a cărui etichetă este 110.

Această variantă este numită enunț *GO TO calculat* sau *GO TO extins*.

2.7.2. **Enunțul IF.** Formele general acceptate ale enunțului IF sînt:

(a) IF < *condiție* > THEN < *enunț-1* > ELSE < *enunț-2* >

(b) IF < *condiție* > THEN < *enunț-1* >

și corespund schemelor logice din fig. 2.8.

Forma (b) poate fi privită ca o restricție a formei (a) care are *enunțul vid* pe ramura corespunzătoare valorii fals a condiției.

Condiția poate fi o expresie relațională, o variabilă logică, o expresie logică sau un nume de condiție (în limbajul COBOL). Deoarece enunțurile de pe cele două ramuri se precizează explicit, rezultă că enunțul IF (sau mai exact IF – THEN – ELSE sau IF – THEN) nu face apel la etichete și poate include alte enunțuri, fiind din acest punct de vedere un enunț structurat. Important este faptul că acest enunț asigură ieșirea în același punct, indiferent de ramura pe care se merge, IF fiind o construcție cu o intrare și o ieșire. Acest fapt are un efect direct asupra lizibilității programelor, ceea ce face ca acest enunț să fie preferat în programe față de unele variante ale sale, cum este enunțul IF aritmetic din limbajul FORTRAN.

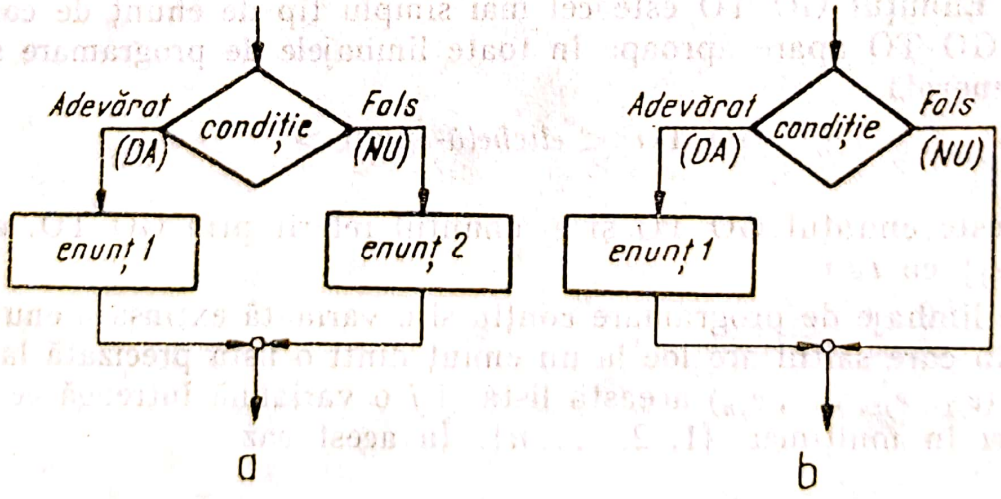



Fig. 2.8. Structuri alternative.



Scanned with OKEN Scanner

O generalizare interesantă a enunțului IF este enunțul CASE din limbajele ALGOL 68 și PASCAL, care a fost propus în anul 1966 [97] ca o extensie la limbajul ALGOL 60. Enunțul CASE are sintaxa:

< *enunț CASE* > : : = CASE < *expresie-întreagă* > OF

BEGIN < *listă-enunțuri* > END

Dacă se consideră o expresie întreagă $i$ ce poate lua valori întregi cuprinse între 1 și $n$ inclusiv și o listă de $n$ enunțuri $e_1$, $e_2$, ..., $e_n$, atunci următoarea construcție ALGOL:

```
CASE i OF
    BEGIN
        e1 ;
        e2 ;
        .
        .
        .
        en
    END
```

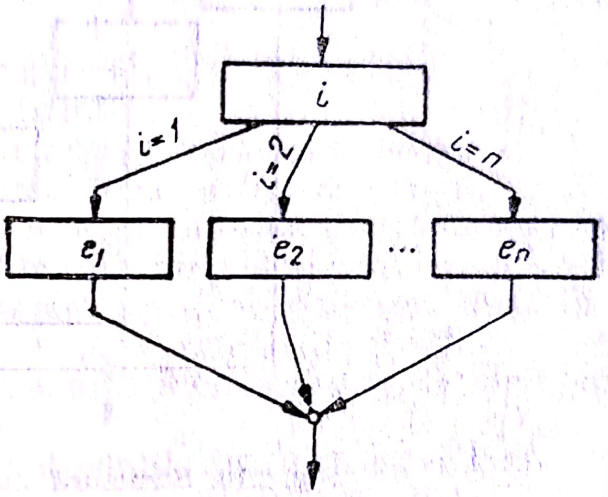



Fig. 2.9. Structură alternativă generalizată

corespunde schemei logice din fig. 2.9. Efectul acestei structuri de control este similar cu cel al lui GO TO extins, cu observația că ramurile conțin chiar enunțurile și nu etichetele acestora.

De asemenea, enunțul CASE dă posibilitatea creării de structuri de programe cu o intrare și cu o ieșire, ceea ce permite simplificarea verificării programelor.

În limbajele de programare în care CASE nu este implementat, o astfel de structură de control se poate simula printr-o succesiune de perechi ELSE – IF în felul următor:

```
IF          i=1  THEN  e1
ELSE  IF    i=2  THEN  e2
.....................................
ELSE  IF    i=n  THEN  en
```

Unii autori [25, 104] indică simularea acestei structuri de control utilizînd enunțul GO TO extins. Este preferabilă însă folosirea de perechi ELSE – IF deoarece se evită utilizarea etichetelor, este mai ușor verificabilă structura simulată și se evită utilizarea enunțului GO TO. Dacă se consideră enunțul CASE ca generalizare a lui IF, înseamnă că fiind

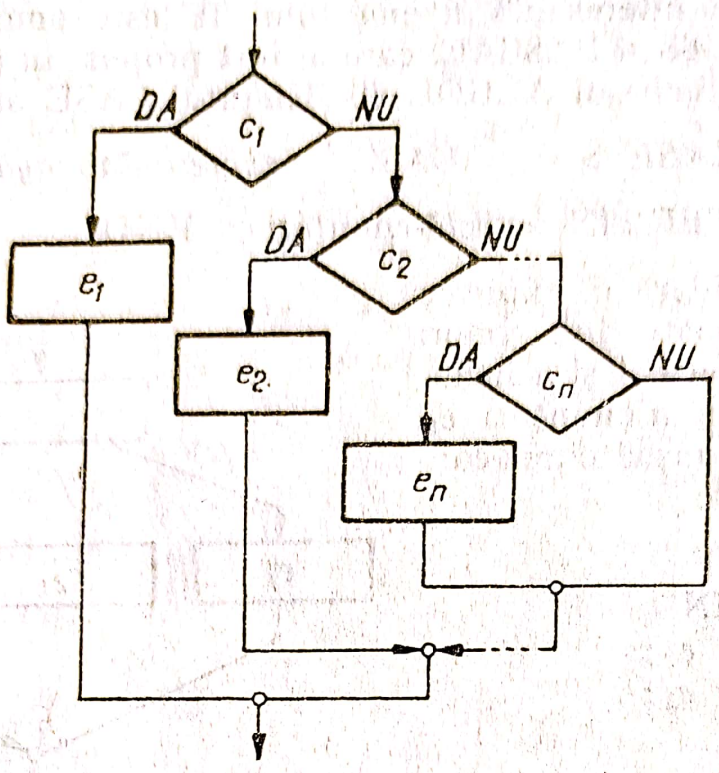



Fig. 2.10. Structură alternativă generalizată exprimată prin structuri alternative.

date $n$ condiţii exclusive $c_1, c_2, \ldots, c_n$, fiecare determinînd o acţiune distinctă descrisă printr-un enunţ $e_i$ şi dacă se notează cu $P(c_i)$ probabilitatea ca $c_i$ să fie adevărată şi se presupune că $P(c_i) \geqslant P(c_j)$ pentru $i<j$, atunci cea mai eficientă secvenţă de perechi ELSE — IF este următoarea :

IF $c_1$ THEN $e_1$
ELSE IF $c_2$ THEN $e_2$
. . . . . .
ELSE IF $c_n$ THEN $e_n$

care corespunde schemei logice din fig. 2.10

2.7.3. **Enunţuri de ciclare.** Ciclul constituie cel mai simplu mod de a executa enunţuri de mai multe ori. Acest lucru este necesar mai ales în cazurile în care aceleaşi acţiuni se execută asupra unei mulţimi de elemente similare (elementele unui tablou, articolele unui fişier etc.). În general, ciclurile pot fi programate utilizînd enunţurile IF şi GO TO. Totuşi, aproape toate limbajele de programare conţin enunţuri speciale pentru a descrie cicluri, de exemplu DO, FOR, PERFORM etc. Existenţa acestor enunţuri permite o programare mai uşoară a ciclurilor cît şi un control mai bun al acestora.



Scanned with OKEN Scanner

Pentru a specifica complet un ciclu trebuie să se descrie :

– structura ciclului, adică enunțurile ce aparțin ciclului (corpul ciclului) ;

– controlul ciclului, adică mijlocul de a controla iterațiile.

Corpul ciclului se definește de obicei prin delimitatori de tip paranteză, specifici fiecărui limbaj în parte. De exemplu, în ALGOL, PL/I, corpul unui ciclu este inclus între un DO și un END. În alte limbaje cum este, de exemplu, COBOL, corpul ciclului este o secvență de program numită *procedură*.

Controlul ciclului presupune, în principiu, următoarele :

– determinarea numărului de iterații ;

– determinarea condiției care va indica terminarea ciclului ;

– determinarea datelor de prelucrat în cursul fiecărei iterații.

Definirea ciclurilor necesită, sub o formă sau alta, evaluarea unei condiții pentru a determina momentul transferului controlului în afara ciclului (ieșirea din ciclu). Este de remarcat faptul că evaluarea condiției se poate face înainte sau după execuția corpului ciclului (fig. 2.11).

În cazul în care evaluarea condiției se face înainte de execuția corpului ciclului, ieșirea din ciclu se asigură :

a) cînd condiția *c* nu mai este îndeplinită (fig. 2.11, *a*), ciclurile fiind construcții de forma

WHILE *c* DO *e*

b) cînd condiția *c* este îndeplinită (fig. 2.11, *b*), ciclurile fiind construcții de forma :

PERFORM *e* UNTIL *c*

Dacă testarea se face după execuția corpului ciclului, ieșirea se asigură cînd condiția *c* este îndeplinită (fig. 2. 11, *c*), ciclurile fiind construcții de forma

DO *e* UNTIL *c*

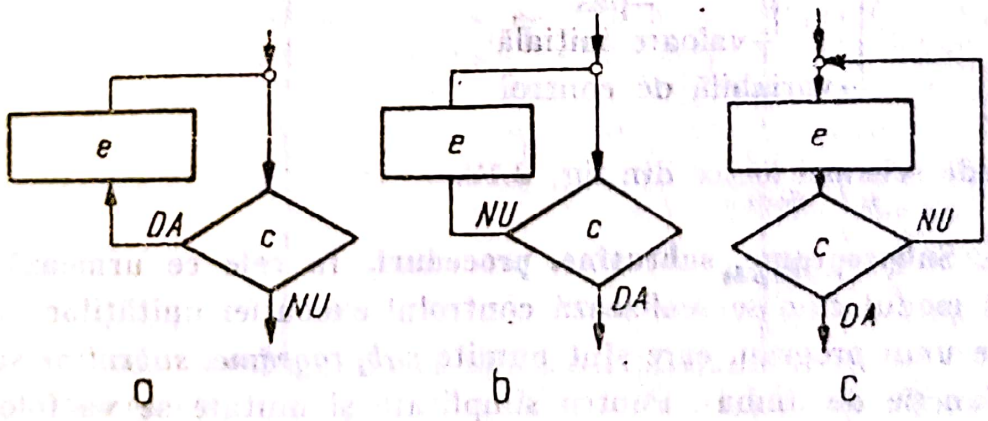



Fig. 2.11. Structuri repetitive.



Scanned with OKEN Scanner

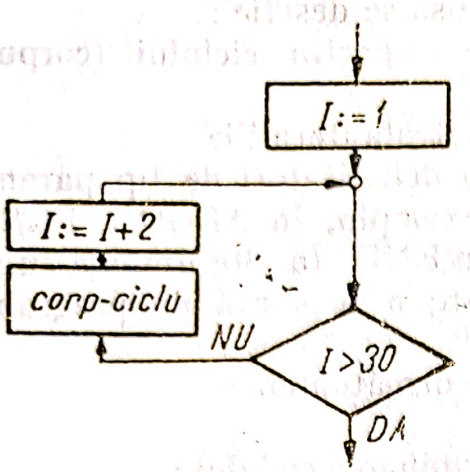



Fig. 2.12. Structură repetitivă cu variabilă de control.

sau

REPEAT *e* UNTIL *c*

De remarcat faptul că, în acest ultim caz, corpul ciclului se execută cel puțin o dată (limbajul FORTRAN este singurul limbaj mai important în care controlul ciclului se face în acest mod). În cazul în care condiția *c* are același tip ca în cazul enunțului IF-THEN-ELSE, atunci ciclul este numit și *ciclu cu condiție*.

O altă formă de control al unui ciclu implică utilizarea unei variabile de control *I* care este incrementată la fiecare execuție a corpului ciclului cu o valoare *p*, numită *pas*. Variabila de control are o valoare inițială $I_0$ și una finală $I_f$. Ieșirea din ciclu se asigură cînd valoarea curentă a variabilei de control depășește valoarea $I_f$. Astfel de cicluri sînt numite *cicluri cu variabilă de control*; valorile curente ale variabilei de control pot fi utilizate pentru selectarea elementelor ce se prelucrează în fiecare pas al ciclului. Ciclurile care se descriu în limbaj FORTRAN sînt exemple tipice de cicluri cu variabilă de control.

Unele limbaje permit definirea de cicluri ce combină cele două metode de control prezentate, evaluarea condiției *c* luînd în considerare valorile variabilei de control. Așa este cazul, de exemplu, al ciclurilor descrise cu enunțul **for** în limbaj ALGOL. Exemplul următor:

**for** *I* : =1 **step** 2 **until** 20 **do** < *corp-ciclu* >

—valoare finală

—pas

—valoare inițială

—variabilă de control

corespunde schemei logice din fig. 2.12.

**2.7.4. Subprograme, subrutine, proceduri.** În cele ce urmează se va prezenta modul cum se realizează controlul execuției unităților independente ale unui program care sînt numite *subprograme*, *subrutine* sau *proceduri*, funcție de limbaj. Pentru simplitate și unitate se va folosi doar noțiunea de *subprogram* ca fiind mai naturală, în raport cu noțiunea de



Scanned with OKEN Scanner

program. Utilizarea subprogramelor în programare este importantă, deoarece permite :

– reducerea efortului de programare prin reutilizarea lor în diferite programe ;

– controlul mai eficace al programelor, fiecare subprogram permiţînd definirea unor nume locale, a unei noi alocări a memoriei, localizarea mai rapidă a erorilor etc. ;

– extensia limbajelor prin definirea de noi enunţuri, noi funcţii intrinseci etc. ;

– modularizarea programelor, deci posibilitatea de a elabora pe părţi şi în echipe sisteme complexe (sisteme de operare, compilatoare, pachete de programe etc.) ; altfel spus, permite realizarea de *programe modulare.*

La abordarea problemei subprogramelor trebuie să se facă distincţie între două aspecte şi anume definirea subprogramelor şi activarea subprogramelor. *Definirea subprogramelor* vizează modul în care subprogramele se alcătuiesc de către programator şi se transmit ca intrări pentru compilator (fig. 2.13). *Activarea subprogramelor* vizează subprogramele în timpul execuţiei sau, altfel spus, comportarea subprogramelor atunci cînd sînt invocate (sau apelate). Această distincţie este mai puţin impor-

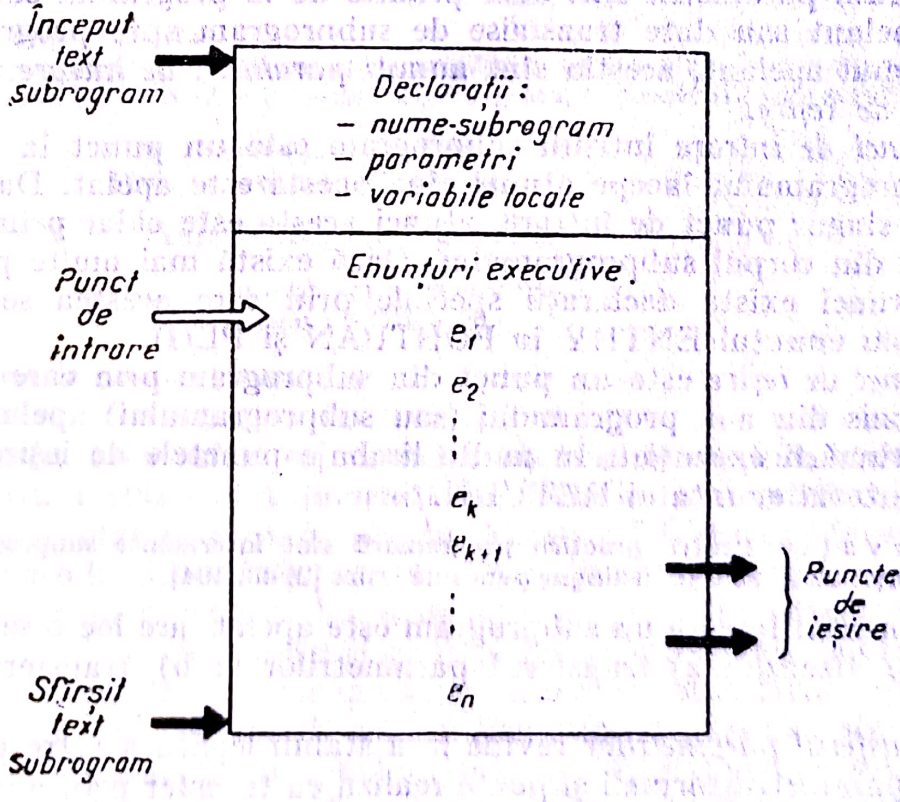



Fig. 2.13. Structura generală a unui subprogram.

tantă în cazul unei structuri de control simple, de tip „call-return", însă devine foarte importantă în cazul în care se consideră tipuri de subprograme mai complexe ca subprograme recursive, corutine, sarcini etc.

Textul unui subprogram (sau corpul subprogramului) este delimitat în mod obișnuit prin enunțuri de tip paranteză, specifice fiecărui limbaj în parte. Corpul subprogramului conține de obicei două categorii de enunțuri :

a) *declarații*, care definesc numele subprogramului, tipul acestuia, parametrii, variabilele locale etc. ;

b) *enunțuri executive*, care descriu operațiile ce se vor executa în timpul apelării subprogramului.

*Parametrii* sînt date (date elementare sau structuri de date) care se transmit unui subprogram în momentul apelării acestuia, sau date care se obțin ca rezultat al execuției subprogramului și se transmit subprogramului (programului) apelant. Deci, fiecărui apel al unui subprogram i se asociază o mulțime de date (în unele cazuri această mulțime poate fi vidă). În momentul definirii unui subprogram, parametrii sînt definiți doar potențial, din care cauză aceștia sînt numiți și *parametri formali* (sau parametri în unele limbaje). În momentul activării unui subprogram ei sînt transmiși efectiv și în această ipoteză sînt numiți *parametri actuali* (sau argumente).

După cum parametrii sînt date primite de la programul sau subprogramul apelant sau date transmise de subprogram spre programul sau subprogramul apelant, aceștia sînt numiți *parametri de intrare*, respectiv *parametri de ieșire*.

Un *punct de intrare* într-un subprogram este un punct la care execuția subprogramului începe atunci cînd acesta este apelat. Dacă există numai un singur punct de intrare, atunci acesta este chiar primul enunț executabil din corpul subprogramului. Dacă există mai multe puncte de intrare, atunci există declarații speciale prin care acestea se definesc (de exemplu enunțul ENTRY în FORTRAN și PL/I).

Un *punct de ieșire* este un punct din subprogram prin care controlul este transmis din nou programului (sau subprogramului) apelant în vederea continuării execuției. În multe limbaje punctele de ieșire se definesc cu ajutorul enunțului RETURN.

Observație. Pentru practica programării sînt interesante subprogramele cu un singur punct de intrare și un singur punct de ieșire [2, 43, 104].

În momentul în care un subprogram este apelat, are loc o secvență de operații ce vizează : a) transferul parametrilor și b) transferul controlului.

a) *Transferul parametrilor* revine la a stabili legătura între parametrii actuali și parametrii formali și poate realiza ca transfer prin nume, transfer prin valori sau transfer prin adrese.



Scanned with OKEN Scanner

1) *Transfer prin nume.* În acest caz în textul subprogramului fiecare parametru formal este înlocuit cu numele parametrului actual transmis, noul text obținut fiind apoi executat. Această tehnică a fost introdusă inițial în ALGOL 60 ([52], p. 12).

**Exemplu.** Fie următoarea procedură ALGOL :

```
procedure POLAR (X, Y, RO, FI) ; real X, Y, RO, FI ;
    begin
        RO : = sqrt (X ↑ 2+Y ↑ 2) ;
        FI : =arctan (Y/X)
    end
```

care calculează coordonatele polare (ρ, φ) funcție de coordonatele carteziene ($x$, $y$) după relațiile $\rho=x^2+y^2$, $\varphi$ = arctg ($y/x$). Apelul procedurii POLAR cu parametrii actuali A, B, R, F, scris sub forma

```
...
POLAR (A, B, R, F)
...
```

poate fi privit ca o înlocuire a enunțului POLAR (A, B, R, F), printr-o copie a textului procedurii, în care aparițiile parametrilor formali X, Y, RO, FI sînt înlocuite cu parametrii actuali A, B, R, F și anume :

```
R : = sqrt (A ↑ 2+B ↑ 2) ;
F : = arctan (B/A)
```

Acest text este apoi executat.

În unele cazuri de apel sînt necesare modificări ale textului. De exemplu, dacă apelul este de forma POLAR (A−B, −D, RO, PSI), textul procedurii este modificat astfel :

```
RO : = sqrt ((A−B) ↑ 2+(−D) ↑ 2) ;
PSI : = arctan (−D)/(A−B))
```

așa încît operațiile să se execute corect și nu sub forma

```
RO : = sqrt (A−B ↑ 2+−D ↑ 2) ,
PSI : = arctan (−D/A−B)
```

cum s-ar părea la prima vedere.

2) *Transfer prin valori* este similar cu transferul prin nume, numai că parametrii formali sînt înlocuiți cu valorile parametrilor actuali și apoi subprogramul este executat. În ALGOL 60 acești parametri sînt declarați într-o listă specială, introdusă prin enunțul *value*.

3) *Transfer prin adrese.* În acest caz programul apelant transmite un tabel ce conține cîte o adresă pentru fiecare parametru actual. Aceste adrese sînt adresele zonelor-memorie alocate parametrilor respectivi.

În cazul în care parametrii actuali sînt expresii, acestea sînt evaluate în programul apelant, iar în tabel se include o adresă reprezentînd adresa zonei de memorie, unde se află valoarea calculată. Această metodă este



Scanned with OKEN Scanner

folosită pentru transferul parametrilor în limbajele FORTRAN, COBOL, PL/I.

Observaţie. În toate cele trei cazuri prezentate corespondenţa între parametrii actuali, furnizaţi sub forma unei liste în enunţul de apel al subprogramului, şi parametrii formali, ce formează o listă în definirea subprogramelor, impune ca lista parametrilor actuali să aibă acelaşi număr de elemente ca şi lista parametrilor formali ; corespondenţa se obţine asociind elementele din cele două liste, în aceeaşi ordine.

b) *Transferul controlului* presupune, în principal, următoarele operaţii :

— salvarea adresei punctului de apel, în aşa fel încît execuţia programului (sau subprogramului) apelant să poată fi reluată după terminarea execuţiei subprogramului apelat ;

— transferul controlului la punctul de intrare în subprogram ;

— transferul controlului la programul apelant, în momentul în care execuţia subprogramului s-a terminat, folosind adresa salvată înainte de intrarea în subprogram.

2.7.5. **Programare structurată.** Se consideră mulţimea $S$ formată din următoarele structuri de control primitive cu o intrare şi o ieşire :

1) operaţii elementare (de exemplu atribuiri, apeluri de proceduri, operaţii de intrare/ieşire etc.) ;

2) structura secvenţială obţinută prin concatenarea a două structuri de control primitive ;

3) structura alternativă de forma „IF $c$ THEN $e_1$ ELSE $e_2$“, unde $c$ este o condiţie, iar $e_1$ şi $e_2$ sînt structuri de control primitive (fig. 2.8, *a*) ;

4) structura repetitivă condiţionată anterior de tip „WHILE $c$ DO $e$“, unde $e$ este o structură de control primitivă, iar $c$ o condiţie (fig. 2.11, *a*).

Un program $P$ este structurat în raport cu mulţimea $S$, dacă este construit numai cu ajutorul structurilor de control primitive cu intrare şi o ieşire aparţinînd lui $S$. Asemenea structuri de control se numesc *structuri Dijkstra* sau *structuri D* [38, 16].

Prin *programare structurată* se înţelege, în general, ansamblul de metode folosite pentru a realiza programe structurate. Programarea structurată se bazează pe o teoremă de structură [11], care afirmă că orice program propriu (un program cu o intrare şi o ieşire) este echivalent cu un program cu structură de control de tip $D$ (este compus numai din structuri de control primitive din $S$).

Avînd în vedere simplitatea structurilor de control primitive, programarea structurată implică o abordare „top-down“ (de la simplu la complex), sau o detaliere prin paşi succesivi („stepwise refinement“), atît la proiectarea programului cît şi la codificarea sa într-un limbaj de programare [96, 25, 38, 49, 110, 100, 16].



Scanned with OKEN Scanner

Avantajul major al acestei metode de programare rezidă în faptul că :

1) se obţin programe cu lizibilitate maximă ;

2) creşte productivitatea programării ;

3) se simplifică testarea şi depanarea programelor ;

4) se poate face o verificare mecanică a programelor ;

5) se obţin programe cu fiabilitate (siguranţă în funcţionare) ridicată [49] ;

6) se obţin programe uşor de întreţinut :

7) se poate realiza o sinteză automată a programelor [39] ;

8) creşte eficienţa globală a programelor.

În practica curentă se realizează programe structurate folosind o clasă mai largă de structuri de control, numită şi clasa structurilor – $D'$ ([25, 38]). Această clasă include, pe lîngă structurile – $D$, următoarele structuri de control primitive :

1) structura alternativă cu o ramură vidă de forma „IF $c$ THEN $e$“, unde $c$ este o condiţie, iar $e$ o structură de control primitivă (fig. 2.8, $b$) ;

2) structura alternativă generalizată de forma „CASE $i$ OF $e_1$ ; $e_2$ ; . . . ; $e_n$“, unde $i$ este o expresie întreagă ($1 \leqslant i \leqslant n$), iar $e_i$ sînt structuri de control primitive (fig. 2.9) ;

3) structura repetitivă condiţionată posterior de tip „REPEAT $e$ UNTIL $c$“, sau „DO $e$ UNTIL $c$“, unde $e$ este o structură de control primitivă, iar $c$ o condiţie (fig. 2.11, $c$).

Observaţii. Structurile din clasa $D'$ constituie ansamblul de structuri de control ale limbajului PASCAL.

Atunci cînd este necesară codificarea într-un limbaj de programare a unui program cu structură de tip $D$ sau $D'$, este necesară definirea unei modalităţi de transcriere a fiecărei structuri de control primitive. În unele cazuri aceasta poate conduce la evitarea totală a instrucţiunii GO TO (cazul limbajului COBOL) sau la o utilizare restrictivă a acesteia (cazul limbajului FORTRAN). În acelaşi timp şi celelalte facilităţi ale limbajului trebuie utilizate conform unor reguli care obişnuit sînt cunoscute sub denumirea de *reguli de stil de programare* (utilizarea lor are în vedere realizarea obiectivelor programării structurate).

# 3

# LIMBAJE DE ASAMBLARE

Un *limbaj de asamblare* este, în general, un limbaj de programare în care, din fiecare instrucţiune se obţine cîte o instrucţiune în limbajul maşină [4, 85]. Cu alte cuvinte, există o corespondenţă biunivocă între instrucţiunile din limbajul maşină şi instrucţiunile unui program scris în limbajul de asamblare. De exemplu, dacă un program în limbaj de asamblare conţine $N$ instrucţiuni (fiecare instrucţiune fiind reprezentată pe cîte o cartelă) şi dacă fiecare instrucţiune din limbajul maşină se reprezintă sub formă de cuvînt (la FELIX C - 256 un cuvînt este o succesiune de 32 biţi), atunci prin asamblarea programului sursă se va obţine un program în limbaj maşină format din $N$ cuvinte.

Instrucţiunile unui limbaj de asamblare care constituie imagini ale instrucţiunilor limbajului maşină se numesc *instrucţiuni maşină*. Pe lîngă această categorie de instrucţiuni, limbajele de asamblare mai conţin instrucţiuni pentru controlul procesului de asamblare (numite *directive* sau *instrucţiuni de asamblare*) şi instrucţiuni care prin asamblare conduc la o succesiune de instrucţiuni din limbajul maşină (numite *macroinstrucţiuni*).

Programele sînt succesiuni finite de instrucţiuni (directive, instrucţiuni maşină, macroinstrucţiuni). La rîndul lor, instrucţiunile au o structură fixă, de obicei compusă din patru părţi (fig. 3.1) : partea de identificare, partea de comandă, partea de argumente şi partea de comentarii.

*Partea de identificare* conţine, în general, un simbol (etichetă) care se asociază unei zone memorie, simbol care permite referirea informaţiei pe care aceasta o conţine (A, B, C, E, CEXP sînt exemple de etichete).

*Partea de comandă* conţine o abreviere simbolică (numită şi *mnemonic*) ce indică tipul instrucţiunii (LD4, L – încărcare cuvînt, AD4, A – adunare cuvînt, ST4, ST – memorare cuvînt, DATA,4,4 şi DC – rezervarea de cuvinte memorie pentru variabile şi iniţializarea lor, constituie exemple de mnemonice).



Scanned with OKEN Scanner

| Parte identificare | Parte comandă | Parte argument | Parte comentarii |
|---|---|---|---|
| CEXP | LD4,1 | A | ÎNCARCĂ A ÎN REGISTRUL 1 |
| | AD4,1 | B | ADUNĂ B LA REGISTRUL 1 |
| | AD4,1 | C | ADUNĂ C LA REGISTRUL 1 |
| | ST4,1 | E | MEMOREAZĂ SUMA LA E |
| | . | | |
| | . | | |
| | . | | |
| A | DATA,4,4 | 5 | REZERVARE CUVÎNT INIȚIALIZAT CU 5 |
| B | DATA,4,4 | 50 | |
| C | DATA,4,4 | 500 | |
| E | DATA,4,4 | 0 | |
| | | a | |
| CEXP | L | 1,A | ÎNCARCĂ A ÎN REGISTRUL 1 |
| | A | 1,B | ADUNĂ B LA REGISTRUL 1 |
| | A | 1,C | ADUNĂ C LA REGISTRUL 1 |
| | ST | 1,E | MEMOREAZĂ SUMA LA E |
| | . | | |
| | . | | |
| | . | | |
| A | DC | F'5' | |
| B | DC | F'50' | |
| C | DC | F'500' | |
| E | DC | 0 | |
| | | b | |

Fig. 3.1. Calculul expresiei E=A+B+C :
a — în limbaj de asamblare pentru FELIX C-256; b — în limbaj de asamblare pentru EMB 370.

*Partea de argumente* este folosită, de exemplu, pentru a specifica adresele operanzilor și registrele folosite pentru executarea operațiilor în cazul instrucțiunilor mașină.

*Partea de comentarii* constituie modalitatea prin care programatorii pot da explicații ajutătoare asupra modului cum lucrează programul.



Scanned with OKEN Scanner

(Un program în limbaj de asamblare fără o astfel de documentare este în general incomprehensibil tuturor programatorilor, cu excepţia autorului, iar după o perioadă de timp de la scrierea sa un program fără comentarii poate deveni greu de înţeles chiar şi pentru autor).

## 3.1. LIMBAJUL ASSIRIS

ASSIRIS este limbajul de asamblare al sistemului de calcul FELIX C–256 (şi IRIS–50) şi este format din mulţimea tuturor programelor ASSIRIS [5, 32, 45, 46, 82, 86, 124]. Programele ASSIRIS sînt unităţile de prelucrare ale asamblorului ASSIRIS. Procesul de asamblare se desfăşoară în doi paşi şi presupune :

– analiza sintactică a programelor ;

– generarea programului obiect, sub forma unor unităţi independente numite *module*.

3.1.1. **Vocabularul limbajului** este o mulţime finită compusă din trei tipuri de elemente : caractere de bază, simboluri, constante.

a) *Caracterele de bază* pot fi : caractere alfabetice literele de la A la Z şi %), caractere numerice (cifrele 0, 1, 2, 3, 4, 5, 6, 7, 8, 9) şi caracterele speciale (+, –, *, /, ., :, virgula (,), apostroful ′ şi caracterul „spaţiu“).

*Tabelul 3.1*

**Tipuri de constante ASSIRIS**

| Tip constantă | Exemple | Semnificaţie | Mod de reprezentare |
|---|---|---|---|
| Întreagă | I′125′ ; 125 | +125 | Cod complementar |
| | I′—125′ ; —125 | —125 | |
| Hexazecimală | X′AB543′ | $AB543_{16}$ | |
| | S′12C′ | $126C_{16}$ | |
| Zecimală | D′475′ ; D′+475′ | +475 | Zecimal împachetat |
| | D′—150′ ; D′150—′ | —150 | |
| Virgulă fixă | FX′—0.150′ | —0,150 | Cod complementar (cuvînt) |
| | FX′7 45E—2′ | $7{,}45 \cdot 10^{-2}$ | |
| | FX′7.45E—2B1′ | $7{,}45 \cdot 10^{-2} \cdot 2^{-10}$ | |
| Virgulă mobilă precizie simplă | FS′150′ ; FS′+150.′ | 150 | Virgulă mobilă (cuvînt) |
| | FS′—4.75E10′ | $-4{,}75 \cdot 10^{10}$ | |
| Virgulă mobilă precizie dublă | FL′150′ ; FL′+150.′ | 150 | Virgulă mobilă (cuvînt dublu) |
| | FL′—4.75E10′ | $-4{,}75 \cdot 10^{10}$ | |
| Şir caractere | 'ALFA' ; C'ALFA' | ALFA | Cod EBCIDIC |
| | 'AC''BD' | AC'BD | |

b) *Simbolul* este orice succesiune de maximum opt caractere alfabetice sau numerice, primul caracter fiind întotdeauna o literă sau %.

c) *Constantele* se folosesc pentru introducerea de numere sau şiruri de caractere direct în textul obiect al programelor. Principalele tipuri de constante sînt prezentate în tabelul 3.1.

3.1.2. **Structura programelor.** Prin *secţiune* ASSIRIS se va înţelege o succesiune de instrucţiuni, prima fiind o directivă CSECT (Control SECTion) sau o directivă DSECT (Dummy SECTion) şi care nu conţine alte instrucţiuni CSECT sau DSECT. În cazul în care o secţiune se încheie cu o directivă END, aceasta se va numi secţiune END.

Un *program* ASSIRIS poate fi : o secţiune END sau o listă de secţiuni urmată de o secţiune END.

În cadrul unui program se pot întîlni următoarele tipuri de secţiuni :

a) *secţiuni de control*, care încep cu o directivă CSECT, ele putînd fi *secţiuni de program*, *de date* şi *de date comune* ;

b) *secţiuni fictive*, care încep cu o directivă DSECT.

Figura 3.2 constituie un program ASSIRIS format dintr-o singură secţiune (o secţiune END). Simbolurile care apar în partea de identificare

```
          CSECT
START     LD4,R0     A      ÎNCARCĂ ÎN REGISTRUL 0 PE A
          LD4,R1     B      ÎNCARCĂ ÎN REGISTRUL 1 PE B
          ST4,R0     B      MEMOREAZĂ REGISTRUL 0 LA B
          ST4,R1     A      MEMOREAZĂ REGISTRUL 1 LA A
          TERM              TERMINĂ EXECUŢIA PROGRAMULUI

A         DATA,4,4   150
B         DATA,4,4   -450
R0        EQU        0
R1        EQU        1
          END        START
```

Fig. 3.2. Program ASSIRIS pentru schimbarea valorilor variabilelor A şi B.

sînt numite şi *simboluri definite*, iar cele care nu sînt nici etichete şi nici mnemonice sînt *simboluri utilizate*.

*Orice simbol utilizat într-un program trebuie să fie şi simbol definit.*

*Reguli de alcătuire a programelor.* a) Secţiunile de control de acelaşi tip trebuie să fie grupate una după alta.

b) Secţiunea END trebuie să fie întotdeauna ultima secţiune dintr-un program ASSIRIS.



Scanned with OKEN Scanner

b) *Simbolul* este orice succesiune de maximum opt caractere alfabetice sau numerice, primul caracter fiind întotdeauna o literă sau %.

c) *Constantele* se folosesc pentru introducerea de numere sau şiruri de caractere direct în textul obiect al programelor. Principalele tipuri de constante sînt prezentate în tabelul 3.1.

3.1.2. **Structura programelor.** Prin *secţiune* ASSIRIS se va înţelege o succesiune de instrucţiuni, prima fiind o directivă CSECT (Control SECTion) sau o directivă DSECT (Dummy SECTion) şi care nu conţine alte instrucţiuni CSECT sau DSECT. În cazul în care o secţiune se încheie cu o directivă END, aceasta se va numi secţiune END.

Un *program* ASSIRIS poate fi : o secţiune END sau o listă de secţiuni urmată de o secţiune END.

În cadrul unui program se pot întîlni următoarele tipuri de secţiuni :

a) *secţiuni de control*, care încep cu o directivă CSECT, ele putînd fi *secţiuni de program*, *de date* şi *de date comune* ;

b) *secţiuni fictive*, care încep cu o directivă DSECT.

Figura 3.2 constituie un program ASSIRIS format dintr-o singură secţiune (o secţiune END). Simbolurile care apar în partea de identificare

```
          CSECT
START     LD4,R0     A      ÎNCARCĂ ÎN REGISTRUL 0 PE A
          LD4,R1     B      ÎNCARCĂ ÎN REGISTRUL 1 PE B
          ST4,R0     B      MEMOREAZĂ REGISTRUL 0 LA B
          ST4,R1     A      MEMOREAZĂ REGISTRUL 1 LA A
          TERM              TERMINĂ EXECUŢIA PROGRAMULUI
A         DATA,4,4   150
B         DATA,4,4   -450
R0        EQU        0
R1        EQU        1
          END        START
```

Fig. 3.2. Program ASSIRIS pentru schimbarea valorilor variabilelor A şi B.

sînt numite şi *simboluri definite*, iar cele care nu sînt nici etichete şi nici mnemonice sînt *simboluri utilizate*.

*Orice simbol utilizat într-un program trebuie să fie şi simbol definit.*

*Reguli de alcătuire a programelor.* a) Secţiunile de control de acelaşi tip trebuie să fie grupate una după alta.

b) Secţiunea END trebuie să fie întotdeauna ultima secţiune dintr-un program ASSIRIS.



Scanned with OKEN Scanner

c) Secţiunile fictive pot apărea oriunde în cadrul programului poziţia lor neafectînd semnificaţia acestuia.

d) În mod obişnuit simbolurile definite într-o secţiune pot fi utilizate numai în cadrul acesteia, fiind numite şi *simboluri locale.*

e) Dacă este necesar ca simbolurile definite într-o secţiune să fie utilizate şi în alte secţiuni, atunci acestea trebuie declarate ca *definiri externe* în secţiunile unde sînt definite şi ca *referiri externe* în secţiunile unde sînt utilizate.

f) Simbolurile definite într-o secţiune fictivă pot fi utilizate fără nici o declarare suplimentară în orice secţiune a programului ASSIRIS care conţine secţiunea fictivă.

3.1.3. **Reprezentarea pe suport a programelor.** 1) Fiecare instrucţiune ASSIRIS se reprezintă pe una sau mai multe cartele succesive.

2) Cele patru părţi ale instrucţiunilor se separă prin unul sau mai multe spaţii succesive.

3) După terminarea părţii de argumente a fiecărei instrucţiuni, pînă în coloana 72, se pot reprezenta comentarii ale programatorului.

4) În program se pot insera şi linii speciale pentru comentarii prin cartele ce conţin * în prima coloană.

5) În cazul în care o instrucţiune trebuie continuată pe o altă cartelă, se pune * în coloana 72 şi se continuă din coloana 22 a cartelei următoare.

3.1.4. **Expresii.** Expresiile ASSIRIS sînt unităţi sintactice ce intră în componenţa instrucţiunilor. În programare se întîlnesc următoarele *tipuri de expresii* :

a) *Expresii aritmetice,* de exemplu : ALFA+2, X+Y−Z ; X+4 * 12 etc., unde ALFA, X, Y, Z sînt simboluri ASSIRIS.

b) *Expresii de tip adresă* : de exemplu : * ALFA ; * 13. ALFA ; * ALFA, 7 etc., unde ALFA este un simbol ASSIRIS.

c) Expresii de tip RA, BA, SG, SE, SD, de exemplu : RA'ALFA' ; BA'ALFA' ; SG'SEG1' ; SD'%' ; SE'X'.

d) *Expresii literale,* de exemplu : L (−256) ; L ('ALFA')

e) *Expresii intrinseci,* de exemplu : AF (I+1) ; CF (2) ; AFA (1).

f) *Expresii de asignare,* de exemplu : RSC:80 ; PRM:OUT.

g) *Expresii idex,* de exemplu : *1 ; *2 ; A1 ; B3.

Dintre aceste tipuri de expresii cele mai răspîndite sînt *expresiile aritmetice.* Expresia aritmetică poate fi : a) un termen ; b) un termen precedat de semnele + sau − ; c) o expresie aritmetică urmată de un operator şi de un termen.

*Operator* poate fi unul din caracterele : +, −, *, /.

*Termen* poate fi un simbol ASSIRIS, caracterul * (semnificînd valoarea curentă a contorului de locaţii), o constantă întreagă, o constantă



Scanned with OKEN Scanner

hexazecimală, o expresie de tip RA, BA, SG, SE sau SD și o expresie intrinsecă.

Se remarcă ușor că expresiile ASSIRIS sînt expresii fără paranteze, din care cauză evaluarea lor se face la fel ca a expresiilor algebrice fără paranteze. Pentru *evaluarea unei expresii aritmetice* asamblorul asociază fiecărui termen al acesteia un număr întreg reprezentat în cod complementar sub formă de cuvînt și apoi evaluarea se face ținînd seama de prioritatea relativă între operatori. La împărțire restul este neglijat, iar dacă valoarea expresiei depășește un cuvînt, atunci rezultatul se trunchiază.

3.1.5. **Instrucțiuni de asamblare (directive).** În principiu acestea servesc pentru controlul procesului de asamblare, pentru rezervări de zone memorie pentru variabile și eventual inițializarea acestora etc. Principalele directive ASSIRIS sînt prezentate în tabelul 3.2.

*Tabelul 3.2*

**Principalele directive din limbajul** ASSIRIS (*et* — etichetă, *s* — simbol reprezentînd adresa primei instrucțiuni de executat în program)

| Directivă | Format | Funcție |
|---|---|---|
| CSECT | [*et*] CSECT [P] | Definește o secțiune de program |
| | [*et*] CSECT D | Definește o secțiune de date |
| | [*et*] CSECT C | Definește o secțiune de date comune |
| END | END [*s*] | Definește sfîrșitul textului unui program ASSIRIS |
| DEF | [*et*] DEF $s_1,\ldots,s_n$ | Declară simbolurile $s_i$ ca definiri externe |
| REF | [*et*] REF $s_1,\ldots,s_n$ | Declară simbolurile $s_i$ ca referiri externe |
| DATA | [*et*] DATA [,$n_1$[,$n_2$]] $e_1,\ldots,e_n$ | Permite rezervarea unor zone memorie de lungime $n_1$ locații la adrese multiple de $n_2$ și inițializarea acestora cu valorile expresiilor $e_i$ |
| TEXT | [*et*] TEXT *șir* | Permite generarea constantelor de tip șir |
| TEXTC | [*et*] TEXTC *șir* | Permite generarea constantelor de tip șir și a lungimii acestora |
| ORG | [*et*] ORG *e* | Poziționarea contorului de locații la valoarea expresiei *e* |
| RES | [*et*] RES[,*n*] *e* | Rezervarea unei zone-memorie de lungime $n \times e$ la adresă multiplu de $n$ ($n$=1, 2, 4, 8) |
| BOUND | [*et*] BOUND *n* | Cadrarea contorului de locații la multiplu de $n$ ($n$=1, 2, 4, 8) |

3.1.6. **Instrucțiunile mașină** constituie imaginile, în limbaj ASSIRIS, ale instrucțiunilor din limbajul mașină al calculatorului FELIX C-256. Forma generală a unei instrucțiuni mașină este următoarea:

etichetă $m$ [,$r$] [*] [$r_b$.] [$d$] [,$x$]



Scanned with OKEN Scanner

unde : $m$ este un mnemonic, de exemplu AD4, ST4, LD4 etc. ;
$r$ — numărul unui registru general, sau o valoare cuprinsă între 0 și 15 ;
$r_b$ — numărul unui registru general folosit ca registru de bază (de la 9 la 15) ;
$d$ — deplasarea (expresie aritmetică cu valori între 0 și 65535) ;
$x$ — numărul unui registru general folosit ca registru index ; caracterul * indică o adresare indirectă.

Principalele tipuri de instrucțiuni mașină din limbajul ASSIRIS sînt :

— instrucțiuni de încărcare și memorare ;
— instrucțiuni de calcul logic ;
— instrucțiuni de decalare ;
— instrucțiuni de salt ;
— instrucțiuni de calcul binar ;
— instrucțiuni de calcul în virgulă mobilă ;
— instrucțiuni de calcul zecimal ;
— instrucțiuni asupra șirurilor de baiți ;
— instrucțiuni de intrare/ieșire.

*Tabelul 3.3*

**Instrucțiunile mașină ale limbajului ASSIRIS**

| Mnemonic | Cod operație | Semnificația | Sintaxa | Principiul de execuție |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| AD4 | 1E | Adunare cuvînt | AD4,r *a, x | $(R_{0-31}) \dot{+} (Y ; 4) \rightarrow R_{0-31}$ |
| AD4I | 27 | Adunare imediată cuvînt | AD4i, r *a, x | $(R_{0-31}) \dot{+} Y_{8-31} \rightarrow R_{0-31}$ |
| AD8 | 1F | Adunare cuvînt dublu | AD8, r *a, x | $(R_{0-63}) \dot{+} (Y ; 8) \rightarrow R_{0-63}$ |
| ADD | 61 | Adunare zecimală | ADD,0,l a<br>ADD,r *a, x | |
| ADF4 | 6C | Adunare în virgulă mobilă simplă precizie | ADF4, r *a, x | $(R_{0-31}) \overset{\circ}{+} (Y ; 4) \rightarrow R_{0-31}$ |
| ADF8 | 7C | Adunare în virgulă mobilă precizie dublă | ADF8, r *a, x | $(R_{0-63}) \overset{\circ}{+} (Y ; 8) \rightarrow R_{0-63}$ |
| ADH2 | 1D | Adunare semicuvînt stînga | ADH2, r *a, x | $(R_{0-15}) \dot{+} (Y ; 2) \rightarrow R_{0-15}$ |
| ADL2 | 1C | Adunare semicuvînt dreapta | ADL2, r *a, x | $(R_{0-31}) \dot{+} (Y ; 2) \rightarrow R_{0-31}$ |
| AI0 | 56 | Analiză intrare/ieșire | AI0, r *a, x | |
| ANALD | 3D | Analiză instrucțiune zecimală | ANALD,r *a, x | |
| ANALN | 3C | Analiză instrucțiune normală | ANALN,r *a, x | |



Scanned with OKEN Scanner

*Tabelul 3.3* (continuare)

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| BAL | 39 | Salt cu legătură | BAL, r *a, x | $(SP_{0-31}) \rightarrow R_{0-31}$ ; $Y_{(14-31)} \rightarrow SP_{14-31}$ |
| BCF | 34 | Salt sub condiţie falsă | BCF, n *a, x | |
| BCT | 35 | Salt sub condiţie adevărată | BCT, n *a, x | |
| BDF | 36 | Salt sub condiţie falsă cu decrementare | BDF, r *a, x | |
| BDT | 37 | Salt sub condiţie adevărată cu decrementare | BDT, r *a, x | |
| BRU | 38 | Salt necondiţionat | BRU *a, x | $Y_{(14-31)} \rightarrow SP_{14-31}$ |
| CP1 | 10 | Comparare bait | CP1, r *a, x | $(R_{0-7}) \; ? \; (Y ; 1) \rightarrow Z, C$ |
| CP1I | 24 | Comparare imediată bait | CP1I, r *a, x | $(R_{0-7}) \; ? \; Y_{(24-31)} \rightarrow Z, C$ |
| CP2 | 11 | Comparare semicuvînt | CP2, r *a, x | $(R_{0-15}) + (Y ; 2) \rightarrow Z, S, D, C$ |
| CP4 | 12 | Comparare cuvînt | CP4, r *a, x | $(R_{0-31}) + (Y ; 4) \rightarrow Z, S, D, C$ |
| CP8 | 13 | Comparare cuvînt dublu | CP8, r *a, x | $(R_{0-63}) + (Y ; 8) \rightarrow Z, S, D, C$ |
| CPD | 63 | Comparare zecimală | CPD, 0, l a<br>CPD, r *a, x | |
| CPSL | 4C | Comparare şiruri de baiţi stînga | CPSL, r *a, x | $((R_{14-31}) ; (R_{48-63})) \; ?$ |
| CPSR | 4D | Comparare şiruri de baiţi dreapta | CPSR, r *a, x | $(Y ; (R_{48-63})) \rightarrow Z, C$ |
| CYBL | 4A | Copiere imediată bait stînga | CYBL, r *a, x | $\{Y_{(24-31)} \rightarrow ((R_{14-31}) ; (R_{48-63}))$ |
| CYBR | 4B | Copiere imediată bait dreapta | CYBL, r *a, x | |
| DV2 | 32 | Împărţire semicuvînt | DV2, r *a, x | $(R_{0-31}) \; : \; (Y ; 2) \rightarrow R_{0-15}$ |
| DV4 | 68 | Împărţire cuvînt | DV4, r *a, x | $(R_{0-63}) \; : \; (Y ; 4) \rightarrow R_{0-31}$ |
| DVD | 65 | Împărţire zecimală | DVD, 0, l a<br>DVD, r *a, x | |
| DVF4 | 6F | Împărţire simplă precizie | DVF4, r *a, x | $(R_{0-31}) \; ? \; (Y ; 4) \rightarrow R_{0-31}$ |
| DVF8 | 7F | Împărţire precizie dublă | DVF8, r *a, x | $(R_{0-63}) \; ? \; (Y ; 8) \rightarrow R_{0-63}$ |
| DVU2 | 33 | Împărţire aritmetică semicuvînt | DVU2, r *a, x | $(R_{0-31}) \; : (Y ; 2) \rightarrow R_{0-15}$ |
| E02 | 0D | Sumă modulo-2 semicuvînt | E02, r *a, x | $(R_{0-15}) \oplus (Y ; 2) \rightarrow R_{0-15}$ |
| E04 | 0E | Sumă modulo-2 cuvînt | E04, r *a, x | $(E_{0-31}) \oplus (Y ; 4) \rightarrow R_{0-31}$ |
| EX2 | 01 | Produs logic semicuvînt | EX2, r *a, x | $(R_{0-15}) \wedge (Y ; 2) \rightarrow R_{0-15}$ |
| EX4 | 02 | Produs logic cuvînt | EX4, r *a, x | $(R_{0-31}) \wedge (Y ; 4) \rightarrow R_{0-31}$ |
| EXU | 3A | Salt cu revenire | EXU *a, x | $Y_{(14-31)} \rightarrow SP_{14-31}$ apoi se revine în secvenţă |
| HIO | 53 | Oprire intrare/ieşire | HIO, r *a, x | |
| IC2 | 46 | Incrementare semicuvînt | IC2, n *a, x | $(Y ; 2) \dot{+} n \rightarrow [Y ; 2]$ |
| IC4 | 47 | Incrementare cuvînt | IC4, n *a, x | $(Y ; 4) \dot{+} n \rightarrow [Y ; 4]$ |
| LAS | 0C | Încărcare şi marcaj | LAS, r *a, x | $(Y ; 4) \rightarrow R_{0-31}$ , $1 \rightarrow Y_0$ |



Scanned with OKEN Scanner

*Tabelul 3.3* (continuare)

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| LD1 | 18 | Încărcare bait | LD1, r *a, x | $(Y ; 1) \rightarrow R_{0-7}$ |
| LD1I | 28 | Încărcare imediată bait | LD1I, r *a, x | $Y_{(24-31)} \rightarrow R_{0-7}$ |
| LD2I | 29 | Încărcare imediată semicuvînt | LD2I, r *a, x | $Y_{(16-31)} \rightarrow R_{1-15}$ |
| LD4 | 0A | Încărcare cuvînt | LD4, r *a, x | $(Y ; 4) \rightarrow R_{0-31}$ |
| LD4I | 2A | Încărcare imediată cuvînt | LD4I, r *a, x | $Y_{(8-31)} \rightarrow R_{8-31} ; 0 \rightarrow R_{0-7}$ |
| LD8 | 0B | Încărcare cuvînt dublu | LD8, r *a, x | $(Y ; 8) \rightarrow R_{0-63}$ |
| LDC | 43 | Încărcarea condiţiilor | LDC, n *a, x | $(Y_{4-7}) \xrightarrow{n} SP_{36-39}$ |
| LDC2 | 19 | Încărcare complement semicuvînt | LDC2, r *a, x | $\overline{(Y ; 2)} \rightarrow R_{0-15}$ |
| LDC4 | 1A | Încărcare complement cuvînt | LDC4, r *a, x | $\overline{(Y ; 4)} \rightarrow R_{0-31}$ |
| LDC8 | 1B | Încărcare complement cuvînt dublu | LDC8, r *a, x | $\overline{(Y ; 8)} \rightarrow R_{0-63}$ |
| LDDA | 2B | Încărcare adresă zecimală | LDDA, 0, l a<br>LDDA, r *a, x | |
| LDH2 | 09 | Încărcare semicuvînt stînga | LDH2, r *a, x | $(Y ; 2) \rightarrow R_{0-15}$ |
| LDL2 | 08 | Încărcare semicuvînt dreapta | LDL2, r *a, x | $(Y ; 2) \rightarrow R_{16-31} ; 0 \rightarrow R_{0-15}$ |
| LDM | 2C | Încărcare multiplă | LDM, r *a, x | $(Y ; \alpha) \rightarrow [0 ; \alpha] ; \alpha = 4 (R+1)$ |
| LDS2 | 45 | Încărcare selectivă semicuvînt | LDS2, r *a, x | $(Y ; 2) \xrightarrow{(R32-47)} R_{0-15}$ |
| LDTM | 41 | Încărcare măşti de depăşire | LDTM, n *a, x | $(Y_{0-3}) \xrightarrow{n} SP_{32-35}$ |
| LPK | 5D | Încărcarea cheilor de protecţie | LPK, r *a, x | |
| LPS | 5E | Încărcarea stării program | LPS, n *a, x | $(Y ; 8) \rightarrow SP_{0-63}$ |
| MG2 | 05 | Sumă logică semicuvînt | MG2, r *a, x | $(R_{0-15}) \vee (Y ; 2) \rightarrow R_{0-15}$ |
| MG4 | 06 | Sumă logică cuvînt | MG4, r *a, x | $(R_{0-31}) \vee (Y ; 4) \rightarrow R_{0-31}$ |
| MP2 | 30 | Înmulţire semicuvînt | MP2, r *a, x | $(R_{0-15}) \times (Y ; 2) \rightarrow R_{0-30} ; 0 \rightarrow R_{31}$ |
| MP4 | 69 | Înmulţire cuvînt | MP4, r *a, x | $(R_{0-31}) \times (Y ; 4) \rightarrow R_{0-62} ; 0 \rightarrow R_{63}$ |
| MPD | 64 | Înmulţire zecimală | MPD, 0, l a<br>MPD, r *a, x | |
| MPF4 | 6E | Înmulţire simplă precizie | MPF4, r *a, x | $(R_{0-31}) \overset{\circ}{\times} (Y ; 4) \rightarrow R_{0-31}$ |
| MPF8 | 7E | Înmulţire precizie dublă | MPF8, r *a, x | $(R_{0-63}) \overset{\circ}{\times} (Y ; 8) \rightarrow R_{0-63}$ |
| MPU2 | 31 | Înmulţire aritmetică semicuvînt | MPU2, r *a, x | $(R_{0-15}) \dot{\times} (Y ; 2) \rightarrow R_{0-31}$ |
| MVSL | 48 | Transfer şir de baiţi stînga | MVSL, r *a, x | $\{((R_{14-31}) ; (R_{48-63})) \rightarrow$ |
| MVSR | 49 | Transfer şir se baiţi dreapta | MVSR, r *a, x | $\rightarrow [Y ; (R_{48-63})]\}$ |
| NF4 | 22 | Normalizare în virgulă mobilă | NF4, r *a, x | |



Scanned with OKEN Scanner

*Tabelul 3.3* (continuare)

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| PACK | 66 | Împachetare | PACK, 0, l a<br>PACK, r *a, x | |
| RD | 50 | Citire directă | RD, r *a, x | |
| SB4 | 16 | Scădere cuvînt | SB4, r *a, x | $(R_{0-31}) \dot{-} (Y ; 4) \rightarrow R_{0-31}$ |
| SB4I | 26 | Scădere imediată cuvînt | SB4I, r *a, x | $(R_{0-31}) \dot{-} Y_{(8-31)} \rightarrow R_{0-31}$ |
| SB8 | 17 | Scădere cuvînt dublu | SB8, r *a, x | $(R_{0-63}) \dot{-} (Y ; 8) \rightarrow R_{0-63}$ |
| SBD | 62 | Scădere zecimală | SBD, 0, l a<br>SBD, r *a, x | |
| SBF4 | 6D | Scădere simplă precizie | SBF4, r *a, x | $(R_{0-31}) \overset{\circ}{-} (Y ; 4) \rightarrow R_{0-31}$ |
| SBF8 | 7D | Scădere precizie dublă | SBF8, r *a, x | $(R_{0-63}) \overset{\circ}{-} (Y ; 8) \rightarrow R_{0-63}$ |
| SBH2 | 15 | Scădere semicuvînt stînga | SBH2, r *a, x | $(R_{0-15}) \dot{-} (Y ; 2) \rightarrow R_{0-15}$ |
| SH2 | 20 | Decalare semicuvînt | SH2, r *a, x | $(R_{0-15}) \xrightarrow{d} R_{0-15}$ |
| SH4 | 21 | Decalare cuvînt | SH4, r *a, x | $(R_{0-31}) \xrightarrow{d} R_{0-31}$ |
| SH8 | 6B | Decalare cuvînt dublu | SH8, r *a, x | $(R_{0-63}) \xrightarrow{d} R_{0-63}$ |
| SHD | 5F | Decalare zecimală | SHD, r *a, x | |
| SIO | 52 | Inițiere intrare/ieșire | SIO, r *a, x | |
| ST1 | 58 | Memorare bait | ST1, r *a, x | $(R_{0-7}) \rightarrow [Y ; 1]$ |
| ST4 | 5A | Memorare cuvînt | ST4, r *a, x | $(R_{0-31}) \rightarrow [Y ; 4]$ |
| ST8 | 5B | Memorare cuvînt dublu | ST8, r *a, x | $(R_{0-63}) \rightarrow [Y ; 8]$ |
| STC | 42 | Memorarea condițiilor | STC, n *a, x | $(SP_{36-39}) \xrightarrow{n} Y_{4-7}$ |
| STH2 | 59 | Memorare semicuvînt stînga | STH2, r *a, x | $(R_{0-15}) \rightarrow [Y ; 2]$ |
| STM | 2D | Memorare multiplă | STM, r *a, x | $(O ; \alpha) \rightarrow [Y ; \alpha] ; \alpha = 4(R+1)$ |
| STPA | 5C | Memorare adresă program | SPTA *a, x | $(SP_{0-31}) \rightarrow [Y ; 4]$ |
| STS2 | 44 | Memorare selectivă semicuvînt | STS2, r *a, x | $(R_{0-15}) \xrightarrow{(R32-47)} [Y ; 2]$ |
| STTM | 40 | Memorarea măștilor de depășire | STTM, n *a, x | $(SP_{32-35}) \xrightarrow{n} Y_{0-3}$ |
| TDV | 55 | Test dispozitiv periferic | TDV, r *a, x | |
| TIO | 54 | Test intrare/ieșire | TIO, r *a, x | |
| TRTL | 4E | Traducere și test stînga | TRTL, r *a, x | |
| TRTR | 4F | Traducere și test dreapta | TRTR, r *a, x | |
| UNPK | 67 | Despachetare | UNPK, 0,l a<br>UNPK, r *a, x | |
| WD | 51 | Scriere directă | WD,r *a, x | |
| WT | 57 | Oprire unitate centrală | WT | |
| ZAD | 60 | Transfer zecimal | ZAD, 0 ,l a<br>ZAD, r *a, x | |

În exprimarea principiului de execuție a instrucțiunilor s-au folosit notațiile :

$R_{m-n}$ — pozițiile binare de la m la n ale registrului general R ;

$(R_{m-n})$ — informația înregistrată în pozițiile binare de la m la n ale registrului general R ;



Scanned with OKEN Scanner

SP — registrul dublu pentru starea programului ;
[Y ; n] — zona-memorie de adresă Y și lungime n locații ;
(Y ; n) — informația înregistrată în zona de adresa Y și lungime n ;
Y — adresa calculată din instrucțiune
$Y_{m-n}$ — pozițiile binare de la m la n din zona-memorie de adresă Y ;
$Y_{(m-n)}$ — cifrele binare de la m la n din adresa calculată Y ;
$\rightarrow$ — transfer informație ;
$\xrightarrow{n}$ — transfer selectiv folosind ca mască informația n ;
$\overline{(Y ; n)}$ — complementul față de 2 al informației din zona [Y ; n] ;
$\dot{+}, \dot{-}, \dot{\times}$, — operații aritmetice în cod complementar ;
$\circ\!+, \circ\!-, \circ\!\times, \circ\!:$ — operații aritmetice în virgulă mobilă
$\xrightarrow{d}$ — transfer cu decalare ;
$\oplus$ — sumă modulo-2 ;
$\vee, \wedge$ — produs logic și sumă logică.

3.1.7. **Macroinstrucțiuni.** Acestea sînt instrucțiuni ASSIRIS traduse de asamblor într-o secvență de instrucțiuni din limbajul mașină. Instrucțiunea TERM (fig. 3.2) este un exemplu de macroinstrucțiune.

## 3.2. LIMBAJUL FAST

Limbajul de asamblare FAST [115] este un subset al limbajulu ASSIRIS. Asamblorul FAST se distinge de asamblorul ASSIRIS prin

– acceptă la intrare doar texte sursă (programe sursă) compuse dintr-o singură secțiune de program și una sau mai multe secțiuni fictive ;

– programul obiect este generat în format IMT și este memorat, după generare, în memoria centraiă (în zona rămasă liberă în partiție după încărcarea asamblorului FAST) ;

– lucrează după sistemul „Load and GO" ce constă în încărcarea o singură dată a asamblorului FAST care, la rîndul său, asigură atît asamblarea cît și lansarea și controlul execuției unui șir de programe sursă succesive.

3.2.1. **Particularitățile limbajului.** 1) Se pot utiliza toate directivele ASSIRIS cu excepția directivelor : DEF, REF, SEGREF, ENTREF, USING.

2) Programul generat fiind executat pe calculatoarele de tip FELIX C-256, rezultă că pot fi utilizate toate instrucțiunile mașină din ASSIRIS, inclusiv cele cu mnemonice extinse.

3) Setul de macroinstrucțiuni monitor este limitat doar la EXUP, WAIT, TYPE, ASSIGN, RELEASE, TIME, TERM, ABORT cu observația că argumentele pentru fiecare macroinstrucțiune trebuie încărcate în registrele generale deoarece zona de argumente este neglijată.



Scanned with OKEN Scanner

4) În programele FAST se pot de asemenea utiliza trei fișiere sistem secvențiale (*1, *2, *3) și patru fișiere de organizare nedefinită pe disc magnetic. Pentru acestea din urmă tabelele de descriere și modulele de intrare/ieșire sînt incluse chiar în asamblor. Deci dintre instrucțiunile și macroinstrucțiunile SGF se pot utiliza:

– SFD, GET, PUT pentru cele trei fișiere sistem;

– UFD, READ, WRITE, SEEK, CHW pentru cele patru fișiere nedefinite (și în acest caz argumentele se încarcă în prealabil în registre);

– OPEN și CLOSE pentru ambele categorii.

3.2.2. **Utilizarea asamblorului FAST.** Asamblorul FAST poate asambla și lansa în execuție mai multe programe succesive, după terminarea asamblării sau a execuției fiecărui program asamblorul preluînd imediat controlul. Diversele operații de executat de către asamblor se indică prin cartele cu % în prima coloană, urmate de o comandă începînd din coloana 3. Astfel:

% COMPILE – indică asamblarea programului FAST care urmează imediat;

% RUN [DUMP] – indică execuția programului asamblat anterior, dacă nu au fost semnalate erori în cursul asamblării;

% GO [DUMP] – indică execuția programului asamblat anterior, în orice condiții; opțiunea DUMP determină tipărirea vidajului zonei memorie ocupată de program în caz de sfîrșit normal al acestuia;

% END – indică sfîrșitul utilizării asamblorului FAST și preluarea controlului de către monitorul de înlănțuire.

În cazul în care un program în execuție utilizează un fișier pe disc magnetic, acesta se descrie folosind cartele de forma % ASSIGN și % LABEL înainte de % RUN sau % GO. Pentru contabilizarea operațiilor efectuate este necesară definirea utilizatorilor. Pentru aceasta se folosește cartela

% USER <*nume-utilizator*>, <*număr-cont*>

**Exemplu de utilizare**

```
.       JOB ...
.       COMPILE FAST
% USER POPESCU, 1120
% COMPILE
  –
  –   } program FAST
  –
  –
% RUN DUMP
  –
  –   } date
  –
```

```
% USER IONESCU, 2570
% COMPILE
  _ }
  _ }   program FAST
  _ }
  _ }
% LABEL A, FN : TDISC', AM : ANY, SZ : 20
% ASSIGN A, DV : RD1
% RUN
  _ }
  _ }   date
  _ }
% END
.         EOJ
```

## 3.3. LIMBAJUL LAF-32

Limbajul de asamblare pentru sistemul de calcul FELIX C-32, LAF-32 este un sublimbaj al limbajului ASSIRIS, prezentînd față de acesta o serie de particularități.

1) *Particularități ale constantelor.* În LAF32 se pot utiliza constante întregi, hexazecimale, în virgulă fixă și în virgulă mobilă, simplă precizie și constante de tip C, cu următoarele restricții:

a) *constantele întregi* se scriu numai sub forma

$$\pm a_n a_{n-1} \ldots a_0,$$

unde $a_i$ sînt cifre zecimale;

b) constantele în virgulă fixă, numai sub forma

$$\mathrm{FX}' \pm \cdot r_1 r_2 \ldots r'_m;$$

unde $r_i$ sînt cifre zecimale; în LAF32 aceste constante se numesc *constante subunitare*;

c) constantele în virgulă mobilă simplă precizie sub forma FS'$rE\alpha'$; în LAF32 acestea sînt numite *constante reale.*

2) *Particularități ale expresiilor.* Expresiile LAF32 se reduc doar la expresiile aritmetice, de adresă, de asignare și idex.

Expresiile aritmetice au ca termeni numai simboluri, caracterul * (reprezentînd valoarea curentă a contorului de locații), constante întregi și constante hexazecimale.



Scanned with OKEN Scanner

3) *Particularităţi ale textelor* LAF32. Secţiunile de control în LAF32 se reduc doar la secţiuni de program. Comunicarea între secţiuni se face la fel ca în limbajul ASSIRIS, cu deosebirea că un simbol declarat ca definire externă printr-o directivă DEF poate fi utilizat în orice altă secţiune de control în textul LAF32, fără a fi necesară precizarea lui într-o directivă REF (prin urmare, directiva REF nu apare în limbajul LAF32).

4) *Particularităţi ale procesului de asamblare.* Procesul de asamblare a textelor LAF32 este similar cu cel al textelor ASSIRIS, cu deosebirea că textul obiect generat este direct încărcabil şi executabil. Acesta este memorat în biblioteca utilizator standard existentă pe discul sistem. În acest fel, faza de editare de legături este eliminată, operaţiile corespunzătoare acestei faze fiind realizate de către asamblorul LAF32.

5) *Particularităţi ale directivelor.* Singura restricţie se referă la fraza CSECT, care nu are nici parte de identificare, nici de argumente.

6) *Particularităţi ale instrucţiunilor maşină.* În limbajul LAF32 se pot utiliza toate instrucţiunile-maşină din ASSIRIS cu excepţia celor care urmează : CYBR, CYBL, TRTR, TRTL, DV4, MP4, ADF8, SBF8. MPF8, DVF8, SLA8, SLL8, SLC8, SRA8, SRL8, SRC8.

# 4

# LIMBAJE PENTRU REZOLVAREA PROBLEMELOR TEHNICO-ȘTIINȚIFICE

Printre cele mai reprezentative limbaje de programare, potrivite rezolvării problemelor tehnico-științifice (calcule multe cu volum de informații mic), figurează limbajele FORTRAN (FORmula TRANslator) și ALGOL (ALGOrithmic Language). În capitolul de față se vor prezenta, mai pe larg, aceste două limbaje și, în plus, limbajele FLAG și LISP, ultimele două foarte pe scurt.

## 4.1. LIMBAJUL FORTRAN

Deși limbajul FORTRAN nu a fost primul limbaj din categoria „limbaje de nivel înalt", a fost, totuși, primul limbaj care a permis lucrul cu rutine și a cunoscut o răspîndire largă (fiind implementat pe numeroase sisteme de calcul) pentru rezolvarea de aplicații nu numai de calcul tehnico-științific. Prima versiune implementată datează din perioada 1954–1956. Prin îmbunătățiri succesive s-au obținut diverse variante ale limbajului, cele cu o stabilitate mai mare fiind cunoscute sub denumirile FORTRAN II și FORTRAN IV (FORTRAN).

Ca eveniment demn de semnalat este standardizarea limbajului, existînd astăzi varianta de limbaj FORTRAN standard.

**4.1.1. Simbolurile de bază.** Limbajul FORTRAN are la bază trei tipuri de simboluri (caractere):

< *literă* > : : = A | B | C | D | E | F | G | H | I | J | K | L | M |
N | O | P | Q | R | S | T | U | V | W | X | Y | Z

< *cifră* > : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

< *caracter special* > : : = + | − | * | / | $\sqcup$ | = | ( | ) | , | . | ' | : | $

Simbolul ⎵ reprezintă caracterul spațiu, iar ultimul este simbolul monetar (unele compilatoare nu acceptă utimele două caractere speciale).

4.1.2. **Identificatori, etichete, constante.** La baza definirii instrucțiunilor limbajului stau următoarele concepte sintactice:

$$< \textit{identificator} > ::= < \textit{literă} > \mid < \textit{identificator} > < \textit{literă} > \mid < \textit{identificator} > < \textit{cifră} >$$

$$< \textit{etichetă} > ::= < \textit{cifră diferită de zero} > \mid < \textit{etichetă} > < \textit{cifră} >$$

$$< \textit{constantă} > ::= < \textit{constantă logică} > \mid < \textit{constantă numerică} > \mid < \textit{constantă alfanumerică} >$$

$$< \textit{constantă logică} > ::= .\text{TRUE.} \mid .\text{FALSE.}$$

Constantele numerice FORTRAN sînt numere întregi sau reale în sensul cunoscut, numai că marca zecimală nu este virgula ci punctul. Pe lîngă aceste două tipuri mai apar trei forme de constante numerice. Așadar, dacă:

$$< \textit{număr} > ::= < \textit{număr întreg} > \mid < \textit{număr real} >$$

atunci

$$< \textit{constantă numerică} > ::= < \textit{număr} > \mid < \textit{număr} > \text{E} < \textit{număr întreg} > \mid$$

$$< \textit{număr} > \{\pm\} < \textit{număr întreg fără semn} > \mid < \textit{număr} > \text{D} < \textit{număr întreg} >$$

Dacă

$$< \textit{șir de caractere} > ::= < \textit{caracter} > \mid < \textit{șir de caractere} > < \textit{caracter} >$$

atunci constantele alfanumerice se definesc în două moduri:

$$< \textit{constantă alfanumerică} > ::= '< \textit{șir de caractere} >' \mid n\text{H} < \textit{șir de } n \textit{ caractere} >$$

**Observații. 1.** Limbajul FORTRAN standard admite identificatori cu cel mult șase caractere. Există unele variante de compilatoare care acceptă identificatori cu pînă la opt caractere sau mai multe, în acest ultim caz fiind semnificative numai primele opt (așa este la FELIX C-256).

2. Etichetele sînt formate din cel mult cinci cifre.

3. Ultimele trei forme de constante numerice sînt variante ale scrierii numerelor în virgulă mobilă, litera D marcînd faptul că este vorba de o constantă în dublă precizie (reprezentată în memoria calculatorului pe două cuvinte).

4. O constantă alfanumerică definită în primul mod (cu apostrof) poate conține $m$ caractere apostrof consecutive dacă se vor insera, la definire, $2m$ caractere apostrof consecutive (a se vedea exemplele date în continuare).

**Exemple**

**Identificatori**

| | | | |
|---|---|---|---|
| AB1 | ANA | MAXIM | V12345 |
| TINAT | VARIAB | MIN | NICAJE |
| MA113 | ALFA | AUX | MARIAN |

**Constante numerice**

| *Scrierea în* FORTRAN | | *Scrierea obișnuită* |
|---|---|---|
| 1976 | +1976 | 1976 |
| −4 | −4 | −4 |
| 1.5E−2 | 15.−3 | $1{,}5 \cdot 10^{-2}$ |
| 0.2D05 | .02D6 | $0{,}2 \cdot 10^{5}$ |
| 0.1+3 | 1.+2 | $0{,}1 \cdot 10^{3}$ |
| −.4−2 | −0.4−2 | $0{,}4 \cdot 10^{-2}$ |

**Constante alfanumerice**

| *Scrierea în* FORTRAN | *Scrierea obișnuită* |
|---|---|
| 'A B C D E F' | A B C D E F |
| 'LIMBAJUL : FORTRAN' | LIMBAJUL : FORTRAN |
| '+−/=' | +−/= |
| 'INTR''UN' | INTR'UN |
| 'A''B''''C' | A'B''C' |
| 5HRABCD | RABCD |
| 6HUTAXYZ | UTAXYZ |

**4.1.3. Variabile.** Există două tipuri de variabile:

< *variabilă* > : : =< *variabilă simplă* > | < *variabilă indexată* >

< *variabilă simplă* > : : =< *identificator* >

< *variabilă indexată*> : : =< *identificator* > (*e*1 [, *e*2] ...)

unde *e*1, *e*2, ... sînt expresii indiciale: întregi fără semn, variabile de tip întreg (vezi paragraful următor), expresii numerice a căror valoare se trunchiază luîndu-se numai partea întreagă. Unele compilatoare admit numai anumite expresii particulare ca indici (FORTRAN FELIX C-256 admite orice expresie numerică, evident, fără numere sau variabile complexe, permițînd maximum șapte indici).



Scanned with OKEN Scanner

**Exemple**

| | | |
|---|---|---|
| TAB | A(4,5) | FI(1000) |
| XY1AB | MATR (3,2,5) | MIMAR |
| MAMA | IND (6) | FORTR (4,2,3,5,6) |

4.1.3.1. *Declararea variabilelor indexate.* Forma generală a instrucţiunii de declarare este :

DIMENSION $<$ *identificator* $>$ ($a1$ [, $a2$] ...)

[, $<$ *identificator* $>$ ($b1$ [, $b2$] ...)] ...

unde prin $a1$, $a2$, ..., $b1$, $b2$, ... s-au notat valorile ultime ale indicilor respectivi, valorile de început ale acestora fiind considerate, implicit, 1 (în ultima variantă de FORTRAN standard se propune declararea ca în ALGOL, limita inferioară putînd fi diferită de 1). Ceea ce se scrie după DIMENSION constituie lista ultimelor componente ale variabilelor indexate declarate. Aici cuvîntul DIMENSION este cuvînt rezervat. În alt context acesta poate avea altă semnificaţie. De exemplu, în varianta FORTRAN FELIX G-256, se poate scrie :

DIMENSION =X+SQRT (A)/Y −2

ceea ce înseamnă că este vorba de variabila DIMENSIO, pentru compilator fiind semnificative numai primele opt caractere.

Rolul instrucţiunii DIMENSION este acela de a pune în evidenţă variabilele indexate (nu şi tipul acestora), ceea ce îi permite compilatorului să rezerve, în memoria centrală, adrese consecutive pentru componente. Cînd variabila are mai mulţi indici, ordinea de variaţie a acestora este de la stînga la dreapta. Aceasta revine, de exemplu în cazul a doi indici, la rezervarea spaţiului de memorie pe coloane (coloana 1, coloana 2, ...).

Tipul variabilelor declarate prin DIMENSION (fără alte precizări) rezultă implicit. Cele al căror nume începe cu una din literele I, J, K, L, M, N sînt de tip întreg, iar restul sînt de tip real.

**Exemple**

DIMENSION IA(4),B (3,5), DELTA (100,200)
DIMENSION M(3), X(3,2)

Pentru exemplul al doilea, variabilele apar în memoria centrală cu adrese consecutive, în ordinea :

M (1),M (2),M (3),X (1,1),X (2,1),X (3,1),X (1,2),X (2,2),X (3,2).

4.1.3.2. *Declaraţii de tip.* În loc de cuvîntul DIMENSION se pot utiliza, respectiv, cuvintele INTEGER, REAL, COMPLEX, LOGICAL, care au ca efect, pe lîngă declararea eventualelor variabile indexate,



Scanned with OKEN Scanner

precizarea tipului acestora (respectiv întreg, real, complex, logic), toate componentele avînd tipul specificat. Într-o declarație DIMENSION apar numai variabile indexate, în timp ce declarațiile de tip pot conține și variabile neindexate.

Dacă se declară variabilele indexate prin DIMENSION și tipul variabilelor nu este cel implicit, atunci indentificatorii variabilelor vor apărea, obligatoriu, în liste ale declarațiilor de tip.

Cînd în loc de DIMENSION se utilizează DOUBLE PRECISION, se pun în evidență variabilele (indexate sau nu) de tip real, reprezentate în dublă precizie (pentru fiecare variabilă se rezervă cîte un cuvînt dublu de memorie).

Pentru a impune ca variabilele ale căror identificatori încep cu anumite litere să fie considerate, implicit, de un anumit tip, se folosește declarația :

IMPLICIT < *t* ▷ (< *literă* > [, < *literă* >] ...) [, < *t* > (< *literă* > [, < *literă* >])] ...

unde

< *t* > : : = INTEGER | REAL | LOGICAL | COMPLEX | DOUBLE PRECISION

Orice succesiune

(< *literă* > [, < *literă* >] ...)

de litere *consecutive* se poate înlocui cu

(< *prima literă a succesiunii* > – < *ultima literă a succesiunii* >)

Mai există un tip de declarație (neacceptată de varianta actuală a compilatorului FORTRAN FELIX C-256), care este de forma

CHARACTER [* < *lungime de caractere* > [,]] < *listă de variabile* >

prin care se pun în evidență variabilele cu lungimi exprimate în caractere (altele decît patru sau opt caractere-cuvînt, respectiv, cuvînt dublu). În unele cazuri, lungimea se exprimă ca număr întreg fără semn, în altele, ca variabilă de tip întreg, iar în cazul cel mai general se dă ca expresie numerică. Cînd lipsește lungimea, se ia, implicit, lungimea un caracter.

4.1.3.3. *Liste de variabile.* Noțiunea *listă de variabile* (sau pe scurt, *listă*) se definește astfel :

< *listă* > : : =< *variabilă neindexată* > | < *componentă de variabilă indexată* > | < *nume de variabilă indexată* > | < *listă* >, < *listă* >



Scanned with OKEN Scanner

Cînd într-o listă apare un nume de variabilă indexată, înseamnă că sînt incluse toate componentele acesteia.

Dacă între indicii variabilelor dintr-o listă se pot găsi anumite relaţii (o valoare de început, o valoare finală şi o raţie pentru indici), atunci este posibilă scrierea prescurtată a listei, aşa cum se se arată în exemplele care urmează (cînd raţia este 1, se poate omite). Ca limite şi raţii pentru indici se pot utiliza şi variabile de tip întreg.

**Exemple**

| *Forma desfăşurată* | *Forma prescurtată* |
|---|---|
| A(1),A(3),A(5),A(7),A(9), | (A(I),I=1,9,2)<br>| | ↓ raţia<br>| ↓ *valoarea finală a indicelui*<br>↓ *valoarea de început a lui* I |
| X (4), Y (2), X (5), Y (3), X(6), Y(4) | (X(K),Y (K—2),K=4,6) |
| M1(1), T (3), M1 (2), T(6), M1(3), T (9) | (M1 (L), T (3*L), L=1,3) |
| A (1,1), A(2,1), A (1,2), A(2,2) | A sau<br>((A(I,J),I=1,2),J=1,2) |
| A(1,1), A (1,2), A(2,2), A(2,2) | ((A(I,J),J=1,2),I=1,2) |
| X, Y, T(2), T(4), T(8), T(16) | X,Y,(T(2**J),J=1,4) |

4.1.4. **Expresii.** Limbajul FORTRAN operează cu trei tipuri de expresii :

< *expresie* > : : =< *numerică* > | < *relaţională* > | < *logică* >

4.1.4.1. *Expresii numerice.* Expresiile numerice sînt reguli de calcul numeric în care intervin operanzi şi operatori numerici :

< *operator numeric* > : : = + | − | / | * | **

< *operand numeric* > : : =< *constantă numerică* > | < *variabilă numerică* > | < *funcţie numerică* >

cu eventuala utilizare a parantezelor pentru a impune o anume ordine de efectuare a operaţiilor (operatorul * marchează înmulţirea, iar ** ridicarea la putere).

**Exemple**

A+X (3)*X (4)−B/C)**2

SIN (X)+COS (X+Y**2)−SQRT(X−Y)

(prin SIN, COS, SQRT s-au notat, respectiv, funcţiile pentru calcularea sinusului, cosinusului, radicalului de ordinul doi).

În paragraful destinat subprogramelor se va da un tabel cu funcţiile utilizabile.

Observaţii. 1. Priorităţile operaţiilor în evaluarea expresiilor numerice sînt următoarele :

- ridicările la putere ;
- înmulţirile şi împărţirile ;
- adunările şi scăderile.

Cînd două operaţi iconsecutive au aceeaşi prioritate, se efectuează în ordinea stînga-dreapta, exceptînd ridicările la putere, unde ordinea este dreapta-stînga.

2. Altă ordine de efectuare (diferită de cea dată de priorităţi) se poate impune prin utilizarea parantezelor.

3. Înainte de efectuarea unei operaţii se converteşte valoarea operandului de tip mai slab în tipul celuilalt operand, tipul rezultatului evaluării unei expresii fiind acela al operandului cu tipul cel mai tare (ordinea tăriei tipurilor, de la cel mai slab la cel mai tare, este: întreg, real simplă precizie, real dublă precizie, complex simplă precizie, complex dublă precizie).

4.1.4.2. *Expresii relaţionale.* Operatorii relaţionali FORTRAN sînt:

< *operator relaţional* > : : =.LT. | .LE. | .EQ. | .GE. /.GT. | .NE. care desemnează respectiv relaţiile: < (Less Than), ⩽ (Less Equal), = (Equal), > (Greater), ⩾ (Greater Equal), ≠ (Not Equal). În aceste condiţii:

◀ *expresie relaţională* > : : =< *expresie numerică* ▷ < *operator relaţional* > < *expresie numerică* ▷

Observaţii. 1. În expresiile numerice care apar într-o expresie relaţională nu există operanzi de tip complex.

2. Evaluarea unei expresii relaţionale conduce la valoarea .TRUE. dacă are loc relaţia respectivă între valorile celor două expresii şi .FALSE. în caz contrar.

**Exemple**

```
X.LE.Y                          X(I).NE.0
(AUX-I*K)*T.EQ.ALFA             A(I, J)-L.GT.1
```

4.1.4.3. *Expresii logice.* Ţinînd seama că operatorii logici acceptaţi de FORTRAN sînt

< *operator logic* > : : =.NOT. | .AND. | .OR.

(care desemnează respectiv funcţiile logice negaţie, conjuncţie, disjuncţie) şi că operanzi logici pot fi

◀ *operand logic* ▶ : : =< *constantă logică* ▷ | < *variabilă logică* > | < *expresie relaţională* ▷ | < *funcţie logică* ▷

rezultă că expresiile logice sînt reguli de calcul logic cu operanzi de tipul specificat.

**Exemple**

```
.NOT.X                          X.AND.Y.AND..NOT.Z.OR.ALFA
A+B.GT.1                        .NOT.A.OR.(X+Y)*2.NE.4
```

Observaţii. 1. Ordinea de prioritate a operatorilor logici este .NOT., .AND., .OR. (la priorităţi egale contează ordinea stînga-dreapta).



Scanned with OKEN Scanner

2. Evaluarea unei expresii logice se face începînd cu evaluarea expresiilor relaționale și apoi conform priorităților operatorilor ; utilizînd parantezele, se poate impune altă ordine de efectuare a operațiilor.

4.1.5. **Instrucțiuni de atribuire.** În general, orice instrucțiune FORTRAN are două componente : eticheta (dacă există) și instrucțiunea propriu-zisă. De obicei, eticheta se aliniază la dreapta în cîmpul 1–5 al formularului de programare, iar instrucțiunea se scrie în cîmpul 7–72 (de obicei, din coloana 7). Continuările sînt puse în evidență de prezența oricărui caracter diferit de spațiu și 0 în coloana 6 a formularului. Pentru compilatorul FORTRAN, spațiul nu este semnificativ.

Există următoarele tipuri de atribuiri : numerică, logică, alfanumerică, de tip ASSIGN, de tip DATA.

4.1.5.1. *Instrucțiunile de atribuire numerică* au structura

$$[<e>] <v> = <\text{expresie numerică}>$$

unde *e* este eticheta, iar *v* variabila numerică neindexată sau componentă a unei variabile indexate.

Dacă tipul rezultat în urma evaluării expresiei numerice nu este același cu al variabilei *v* se face conversia tipului valorii expresiei la tipul variabilei.

**Exemple**

```
AUX=4*A-B**I
BT=I*I*X-K/H*(1-B**B)
W=A (I,2*J,K)-X
```

4.1.5.2. *Instrucțiunile de atribuire logice* au forma generală.

$$[<e>] <v> = <\text{expresie logică}>$$

unde *v* este variabilă de tip logic neindexată sau componentă de variabilă indexată. Variabila *v* primește valoarea .TRUE. dacă evaluarea expresiei logice conduce la .TRUE. și .FALSE. în caz contrar.

**Exemple**

```
T=X.AND..NOT.Y
S=.TRUE.
```

4.1.5.3. *Instrucțiunile de atribuire alfanumerică* au forma generală :

$$[<e>] <v> = <\text{expresie alfanumerică}>$$

unde *v* este variabilă neindexată sau componentă de variabilă indexată, iar expresia alfanumerică este constantă alfanumerică, variabilă alfanumerică neindexată, componentă de variabilă alfanumerică indexată.



Scanned with OKEN Scanner

Observaţie. Dacă $m$ este lungimea (în caractere) a lui $v$, iar $n$ lungimea expresiei, atunci există trei situaţii : a) $m=n$ care nu ridică probleme ; b) $m>n$ care conduce la completarea lui $v$, la dreapta, cu $m-n$ spaţii ; c) $m<n$ implică neglijarea ultimelor $n-m$ caractere ale expresiei.

**Exemple**

```
X=AUXI
T=V (I)
```

4.1.5.4. *Instrucţiunea de atribuire* ASSIGN are structura generală

[<*e*>] ASSIGN <*e*> TO < *variabilă de tip întreg*>

unde *e* este etichetă.

Observaţie. Acest tip de instrucţiune se utilizează asociată cu GO TO asignat (impus).

4.1.5.5. *Instrucţiunile de tip* DATA au forma generală

DATA < *var*−1> / <*v*−1> / [[,] <*var*−2> / <*v*−2 > /]...

*var*−1, *var*−2, ... fiind liste de variabile, iar *v*−1, *v*−2, ... liste de valori ce urmează să se atribuie variabilelor. În mod normal lista de valori *v*−1 are atîtea elemente cîte are lista *var*−1, *v*−2 atîtea cîte are *var*−2, ..., existînd concordanţă de tip.

Observaţie. Pentru valori egale se admite scrierea cu factor de multiplicitate (de exemplu, lista /−1, −1, −1, −1, 2, 2, 2, 2/ se poate scrie /4*(−1), 4*2/).

**Exemple**

```
DATA A, B, C, D/.TRUE., 'XYZT', 4., .5/
DATA A (1,1),X,Y,Z/(.5, 4.),1H*,2*5)./
```

4.1.6. **Instrucţiuni de control.** În această categorie sînt cuprinse următoarele instrucţiuni :

- instrucţiunile STOP şi PAUSE ;
- instrucţiunile de tip GO TO ;
- instrucţiunile de tip IF ;
- instrucţiunea DO ;
- instrucţiunile RETURN şi CALL (vezi 4.1.7.2).

4.1.6.1. *Instrucţiunile* STOP *şi* PAUSE au ca rol întreruperea execuţiei unui program, în cazul lui STOP fără posibilitate de continuare, iar în cazul lui PAUSE cu posibilitatea continuării din punctul rămas (printr-un mesaj la consola calculatorului) şi pot fi întîlnite în una din formele :

[< *etichetă* >] STOP [< *întreg fără semn* >]
[< *etichetă* >] STOP ['< *şir de caractere* >']



Scanned with OKEN Scanner

[< *etichetă* >] PAUSE [< *întreg fără semn* >]
[< *etichetă* >] PAUSE ['< *șir de caractere* >']

însă în marea majoritate a situațiilor apar sub forma de STOP, respectiv PAUSE, eventual cu etichetă.

4.1.6.2. *Instrucțiunile de tip* GO TO au rolul de a indica instrucțiunea ce urmează să se execute. Acestea pot fi de trei tipuri : GO TO necondiționat, GO TO calculat și GO TO asignat.

Instrucțiunea GO TO necondiționat este de forma

GO TO <*e*>

și are ca efect executarea instrucțiunii de etichetă *e*.

Structura generală a instrucțiunii GO TO calculat este

[<*e*>] GO TO (<*e*1>[, <*e*2>] ...) [,] < *expresie numerică*>

așa cum este acceptată de varianta standard a limbajului (expresia nu poate conține variabile de tip complex). În majoritatea cazurilor, expresia numerică se reduce la o variabilă de tip întreg (este și cazul compilatorului FORTRAN FELIX C-256).

Un GO TO calculat este echivalent cu un GO TO <*ei*> dacă *i* este valoarea expresiei numerice. Dacă $i<1$ sau $i>n$ ($n$ fiind numărul etichetelor din lista instrucțiunii), atunci GO TO calculat este ineficient trecîndu-se la instrucțiunea următoare.

GO TO asignat are structura :

[<*e*>] GO TO < *variabilă întreagă* > [[,] (<*e*1>[, <*e*2>] ...)]

și are ca efect un GO TO <*ei*> dacă *ei* este una din valorile *e*1, *e*2, ... atribuită variabilei (fie *v* aceasta) cu o instrucțiune ASSIGN :

ASSIGN <*ei*> TO *v*

**Exemple**

```
GO TO 1901
GO TO (4,7,3),I
GO TO (1,9,14,2) J
GO TO M

GO TO N,(14,15,1)
```

```
GO TO 7    dacă I=2
GO TO 2    dacă J=4
GO TO 6    dacă anterior execuției
se efectuează instrucțiunea
ASSIGN 6 TO M
GO TO 15 dacă anterior se efectuează
ASSIGN 15 TO N
```

4.1.6.3. *Instrucțiunile de tip* IF au două forme : IF aritmetic și IF logic. Structurile acestora sînt respectiv :

[<*e*>] IF (< *expresie numerică* >) <*e*1>, <*e*2>, <*e*3>



Scanned with OKEN Scanner

[<e>] IF (< *expresie logică* >) *i*

unde *e*, *e*1, *e*2, *e*3 sînt etichete (expresia numerică nu conține tipul complex) și *i* este o instrucțiune diferită de IF logic, DO, END.

Instrucțiunea IF aritmetic este echivalentă cu GO TO <*e*1> dacă *v*<0, cu GO TO <*e*2> dacă *v*=0 și cu GO TO *e*3 dacă *v*>0 (cu *v* s-a notat valoarea expresiei numerice).

Execuția unui IF logic decurge astfel : dacă valoarea expresiei logice este .TRUE., atunci se execută instrucțiunea *i* și apoi următoarea (lucrurile decurg altfel dacă *i* este la rîndul ei tot o instrucțiune de control), iar în caz contrar (valoarea expresiei este .FALSE.) se execută instrucțiunea ce urmează lui IF logic (este evitată instrucțiunea *i*).

**Exemple**

```
145 IF(X-4)7,7,9          ↔   GO TO 7   dacă X≤4 sau
                              GO TO 9   dacă X>4

    IF (X.GT.0) K=1       ↔   K=1       dacă X>0 ;
                              dacă X≤0 nu se mai efectuează
                              K=1

    IF (X(I).NE.Y(I)) IF (I-5) 1,4,7
    IF (X.EQ.Y) STOP
```

4.1.6.4. *Instrucțiunea* DO este o instrucțiune compusă, utilizată pentru definirea ciclurilor (parcurgerea repetată a unui grup de instrucțiuni) și se definește astfel :

[<*e*>] DO <*e*1> [,] <*id*>=<*vi*>, <*vf*> [, <*r*>]

unde *e* este eticheta opțională (este necesară cînd se atinge acest DO printr-o instrucțiune de tip GO TO sau IF) ; *e*1 este eticheta instrucțiunii care marchează sfîrșitul fizic al lui DO ; *id* este un identificator de variabilă de tip întreg (FORTRAN standard admite și tipurile real și dublă precizie pentru *id*) numită variabila de control a ciclului ; *vi*, *vf* și *r* sînt sau constante întregi fără semn sau variabile de tip întreg, reprezentînd, respectiv, valoarea inițială a variabilei de control (*vi*), valoarea finală a acesteia (*vf*) și rația de variație de la *vi* către *vf* (cînd valoarea rației *r* este 1, se poate omite scrierea acesteia).

Observații. 1. Ca instrucțiune ce marchează sfîrșitul fizic al lui DO nu pot fi utilizate : DO, GO TO, IF aritmetic, RETURN, STOP, END.

2. Se poate include total un DO în alt DO.

3. Două sau mai multe instrucțiuni DO pot avea aceeași instrucțiune de sfîrșit, în acest caz executîndu-se cu prioritate (dintre cele neexecutate) cel mai apropiat DO față de instrucțiunea terminală.

4. Dacă drept linie terminală a unui DO ar fi (dictat de natura problemei ce se rezolvă) instrucțiune interzisă, se ia ca linie terminală a lui DO (cu eticheta *e*1 din



Scanned with OKEN Scanner

definiție) o instrucțiune CONTINUE, cea interzisă devenind penultima instrucțiune a lui DO.

5. Intrarea într-un DO se face prin linia de definiție. Un DO are o ieșire naturală (atingerea primei instrucțiuni de după cea terminală) și ieșiri fortuite (în funcție de anumite condiții).

6. Pentru o desfășurare corectă a calculelor este interzis să se modifice valorile variabilei de control.

7. Domeniul extins al lui DO este constituit din ansamblul unor instrucțiuni atinse prin părăsirea lui cu un GO TO sau un IF după care (în caz că nu s-a atins un STOP) se revine în ciclu (cel puțin pentru a atinge linia terminală care să provoace avansul și controlul variabilei de control).

4.1.7. **Subprograme.** Subprogramele se pot clasifica astfel :

- Subprograme
  - funcție
    - predefinite (intrinseci)
    - definite de programator
  - externe
    - de tip FUNCTION
    - de tip SUBROUTINE
  - de tip BLOCK DATA

Înainte de a trece la prezentarea tipurilor de subprograme, vom da cîteva considerații privind instrucțiunile EQUIVALENCE, COMMON, EXTERNAL, INTRINSIC, PARAMETER.

1) *Instrucțiunea* EQUIVALENCE are structura

EQUIVALENCE (< *listă* >) [, (< *listă* >)] ...

prin *listă* specificîndu-se variabilele declarate echivalente. În felul acesta unele zone ale memoriei pot fi recunoscute sub mai multe nume, ceea ce se traduce prin economie de memorie, în cadrul unui program. Echivalența se referă la loc, nu la valoare. De aceea nu este obligatoriu ca variabilele declarate echivalente să aibă același tip. O variabilă neindexată nu poate fi echivalentă decît cu prima componentă a unei variabile indexate (în listele din EQUIVALENCE a scrie, de exemplu, A(1,1) pentru o matrice A este tot una cu a scrie numai numele matricei, A).

Cînd se lucrează cu cuvinte duble pot să apară greșeli de adresare (unele cuvinte duble să nu apară la adrese multiplu de 8). Pentru evitarea acestor situații se poate profita de faptul că pentru zonele declarate prin EQUIVALENCE (și COMMON) compilatorul rezervă ca adresă de început una multiplu de 8.

Dacă echivalările se fac de așa manieră încît ordinea variabilelor să fie convenabilă (cele de lungime dublă la începutul zonei, apoi cuvintele, semicuvintele), nu vor mai apărea erori de adresare.

2) *Instrucțiunea* COMMON are structura

COMMON [/ [< *nb* >] /] < *listă* > [[,] / [< *nb* >] / < *listă* >] ...



Scanned with OKEN Scanner

unde *nb* sînt nume alese (identificatori) pentru blocurile constituite din listele considerate (nume de blocuri). Cînd *nb* lipsește este vorba de *blocul blanc*. Dacă acesta este chiar primul, pot fi omise cele două bare. Celelalte blocuri se numesc *blocuri etichetate*. Reapariția numelui unui bloc face ca să se extindă lista variabilelor ce-l compun. Blocurile comune etichetate, utilizate de mai multe programe, trebuie să aibă aceeași lungime. Pentru blocul blanc nu se impune această cerință.

Instrucțiunea COMMON face ca unele zone de memorie să fie accesibile mai multor programe, aceste zone putînd fi recunoscute sub nume diferite. Pe lîngă economie de memorie, COMMON facilitează lucrul în echipe, permite segmentarea programelor.

Și la COMMON pot apărea erori de adresare. Pentru evitarea acestora se poate proceda după cum s-a arătat la EQUIVALENCE.

Exemple : 1. Dacă se consideră instrucțiunile

```
INTEGER A (3,4), V (9)
EQUIVALENCE (A (1,2), V (4))
```

rezultă că cele 12 cuvinte ale unei zone de memorie vor fi recunoscute respectiv sub numele A (1,1), A (2,1), A (3,1), A (1,2), A (2,2), ..., A (3,4), iar primele nouă cuvinte și sub numele V (1), V (2), ..., V (9) (fig. 4.1).

| $A_{11}$ | $A_{21}$ | $A_{31}$ | $A_{12}$ | $A_{22}$ | $A_{32}$ | $A_{13}$ | $A_{23}$ | $A_{33}$ | $A_{14}$ | $A_{24}$ | $A_{34}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ | $V_6$ | $V_7$ | $V_8$ | $V_9$ | — | — | — |

Fig. 4.1. Variabile echivalente.

2. Dacă în trei programe P1, P2, P3 se consideră respectiv grupurile de instrucțiuni

```
LOGICAL L
COMPLEX C
COMMON /B1/A, X(5) / /L/B2/I (3)/ /C
```

```
LOGICAL P
COMMON P, E /B1/T (2),Y (4)/B2/K(3)
```

```
INTEGER V(3)
COMMON /B2/V/B1/Y (6)
```

atunci cele trei blocuri (două etichetate B1, B2 și blocul blanc) vor fi utilizate de P1, P2, P3, după cum se arată în fig. 4.2.

Blocul blanc

| L | C | |
|---|---|---|
| P | E | — |
| — | — | — |

în P1

în P2

în P3

Blocul B1

| A | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ |
|---|---|---|---|---|---|
| $T_1$ | $T_2$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ |
| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ |

Blocul B2

| $I_1$ | $I_2$ | $I_3$ |
|---|---|---|
| $K_1$ | $K_2$ | $K_3$ |
| $V_1$ | $V_2$ | $V_3$ |

Fig. 4.2. Variabile declarate în COMMON.



Scanned with OKEN Scanner

Se observă că blocul blanc în primul program a fost constituit mai întîi din L la care s-a adăugat C. În programul P3 blocul blanc nu s-a utilizat.

3) *Instrucțiunile* EXTERNAL, INTRINSIC, PARAMETER. Instrucțiunea EXTERNAL are structura

EXTERNAL < *identificator* > [, < *identificator* >] ...

prin care se specifică nume de proceduri externe ce se vor utiliza ca parametri actuali. Instrucțiunea EXTERNAL este plasată în programul apelant.

Instrucțiunea INTRINSIC are structura

INTRINSIC < *identificator* > [, < *identificator* >] ...

și precizează nume de funcții intrinseci (predefinite), care se vor utiliza ca parametri actuali. Instrucțiunea INTRINSIC apare în programul apelant.

Observație. Nu se pot utiliza ca parametri actuali (nu pot apărea în instrucțiunea INTRINSIC) funcțiile intrinseci MAX0, AMAX1, AMAX0, MAX1, MIN0, MIN1, AMIN0, AMIN1.

Instrucțiunea PARAMETER are structura

PARAMETER < *id* > = < *constantă* > [, < *id* > = < *constantă* >] ...

și se utilizează pentru a da nume unor constante. Tipul numelui ales este cel al constantei utilizate. Nu are efect tipul dat de prima literă a identificatorului.

Observație. Nu toate variantele de compilatoare acceptă INTRINSIC și PARAMETER.

4.1.7.1. *Subprograme funcție.* Sînt de două tipuri: *funcții predefinite (intrinseci)* și *funcții definite de programator (funcții formulă).*

Funcțiile intrinseci sînt prezentate în tabelul 4.1, iar funcțiile formulă se definesc (în cadrul programului în care se utilizează) astfel:

$$f(X1, X2, \ldots, XN) = E$$

unde $f$ este numele funcției (numele este un identificator FORTRAN); $X1, X2, \ldots, XN$ sînt argumentele funcției (se numesc parametri formali), iar $E$ este expresia de definire a funcției și poate să conțină ca operanzi:

- parametri formali;
- constante, exceptînd pe cele alfanumerice;
- variabile, exceptînd pe cele alfanumerice;
- funcții intrinseci;
- funcții definite anterior;
- subprograme externe de tip FUNCTION.



Scanned with OKEN Scanner

*Tabelul 4.1*

**Funcții predefinite**

| Nr. crt. | Funcția | Nr. de arg. | Nume generic | Nume specific | Tipul argument | Tipul funcției | Observații |
|---|---|---|---|---|---|---|---|
| 1 | Conversie | 1 | INT | INT | R | I | INT (3.6)=3 |
| | | | | IFIX | R | I | IFIX (3.6)=3 |
| | | | | IDINT | DP | I | |
| | | | REAL | FLOAT | I | R | FLOAT (2)=2. |
| | | | | SNGL | DP | R | |
| | | | | REAL | C | R | |
| | | | DBLE | DFLOAT | I | DP | |
| | | | — | DBLE | I | DP | |
| | | | — | AIMAG | C | R | |
| | | | — | CONJG | C | C | |
| | | | CMPLX | CMPLX | R | C | CMPLX(2.,0.)=(2.,0.) |
| 2 | Valoare absolută | 1 | ABS | IABS | I | I | IABS (—2)=2 |
| | | | | ABS | R | R | |
| | | | | DABS | DP | DP | |
| | | | | CABS | C | R | |
| 3 | Rădăcina pătrată | 1 | SQRT | SQRT | R | R | SQRT (4.)=2. |
| | | | | DSQRT | DP | DP | |
| | | | | CSQRT | C | C | |
| 4 | Exponențierea cu baza e | 1 | EXP | EXP | R | R | EXP(2.)=$e^2$ |
| | | | | DEXP | DP | DP | |
| | | | | CEXP | C | C | |
| 5 | Logaritm natural | 1 | LOG | ALOG | R | R | |
| | | | | DLOG | DP | DP | |
| | | | | CLOG | C | C | |
| 6 | Logaritm zecimal | 1 | LOG10 | ALOG10 | R | R | ALOG10 (100.)=2. |
| | | | | DLOG10 | DP | DP | |
| 7 | Sinus | 1 | SIN | SIN | R | R | |
| | | | | DSIN | DP | DP | |
| | | | | CSIN | C | C | |
| 8 | Cosinus | 1 | COS | COS | R | R | |
| | | | | DCOS | DP | DP | |
| | | | | CCOS | C | C | |
| 9 | Tangentă | 1 | TAN | TAN | R | R | |
| | | | | DTAN | DP | DP | |
| 10 | Signum (transfer de semn) | 2 | SIGN | ISIGN | I | I | SIGN (A, B) |
| | | | | SIGN | R | R | este \|A\| dacă B>0 |
| | | | | DSIGN | DP | DP | și —\|A\| dacă B<0, |



Scanned with OKEN Scanner

*Tabelul 4.1* (continuare)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 11 | Diferența pozitivă | 2 | DIM<br>DIM | IDIM<br>DIM<br>DDIM | I<br>R<br>DP | I<br>R<br>DP | IDIM (I, J) este I—J dacă I<J și este 0 altfel |
| 12 | Calculul maximului | n | MAX | MAX0<br>AMAX1<br>DMAX1 | I<br>R<br>DP | I<br>R<br>DP | MAX (3—1,2)=3 |
| 13 | Calculul minimului | n | MIN | MIN0<br>AMIN1<br>DMIN1 | I<br>R<br>DP | I<br>R<br>DP | AMIN (4., 3)=3. |
| 14 | Arcsin | 1 | ASIN | ASIN<br>DASIN | R<br>DP | R<br>DP | |
| 15 | Arccos | 1 | ACOS | ACOS<br>DACOS | R<br>DP | R<br>DP | |
| 16 | Arctg | 1 | ATAN | ATAN<br>DTAN | R<br>DP | R<br>DP | ATAN (X) |
| | | 2 | — | ATAN2<br>DTAN2 | R<br>DP | R<br>DP | ATAN2 (X/Y) |
| 17 | Sinus hiperbolic | 1 | SINH | SINH<br>DSINH | R<br>DP | R<br>DP | |
| 18 | Cosinus hiperbolic | 1 | COSH | COSH<br>DCOSH | R<br>DP | R<br>DP | |
| 19 | Tangentă hiperbolică | 1 | TANH | TANH<br>DTANH | R<br>DP | R<br>DP | |

Funcțiile definite pot fi numerice sau logice. Dacă tipul funcției *f* nu rezultă implicit, acesta trebuie explicitat (declarat).

Subprogramele funcție, fie că sînt de tip formulă (definite de programator), fie că sînt intrinseci, se apelează apărînd ca operanzi în expresii de tip corespunzător.

Dacă, de exemplu, se definește

F (X,Y,Z)=X*X+Y*Y+Z*Z

atunci prin instrucțiunea

R=F (A,B,C)+F (3.,A+B+C,−1)

se calculează.

$$R=A^2+B^2+C^2+3.^2+(A+B+C)^2+(-1.)^2$$

Fie funcția logică

$$F(X,Y,Z)=X\wedge\overline{Y}\vee Y\wedge\overline{Z}\vee Z\wedge\overline{X}.$$

O astfel de funcție se va defini și utiliza, de exemplu, astfel :

```
LOGICAL X, Y, Z, R, F
F (X,Y,Z)=X.AND..NOT.Y.OR.Y.AND..NOT.Z.OR.Z.AND.NOT.X
R=F (X,Y,Z)
```

Se mai poate proceda, spre exemplu :

```
LOGICAL X, Y, Z, R, F
F (T,U)=T.AND.U
R=F (X,.NOT.Y).OR.F (Y.,NOT.Z).OR.F (Z,.NOT.X)
```

Fie următorul exemplu :

```
COMPLEX A,B,C,G.SC,PCX,Y,Z,T
SC (X,Y,Z)=X+Y+Z
PC (X,Y,Z)=X*Y×Z
G (X,Y)=X+Y
T=G (SC (A,B.C), PC (CMPLX (3.,2), CONJ (A), (1.,4))
```

unde, dacă

$$A=3+4i\ ;\ B=1-i\ ;\ C=4+2i,$$

se va calcula

$$T=3+4i+1-i+4+2i+(3+2i).(3-4i).(1+4i).$$

O b s e r v a ț i i. 1. O funcție formulă se poate utiliza numai în programul în care a fost definită.

2. Funcțiile formulă nu pot să apară în instrucțiuni EXTERNAL.
3. Funcțiile formulă nu pot fi de tip caracter.
4. Funcția însăși nu poate să apară ca operand în expresia ei de definiție.
5. Ca parametri formali nu se pot lua :
   — funcția însăși ;
   — o expresie care o conține ;
   — o funcție a cărei expresie o conține.
6. Numele dat funcției nu poate fi utilizat ca nume de variabilă.

4.1.7.2. *Subprograme externe.* Acestea pot fi de tip FUNCTION sau de tip SUBROUTINE. Cele de tip FUNCTION se definesc astfel :

$$[<t>]\ \text{FUNCTION}\ <nume>\ [(p\ [,p]\ \ldots)]$$

unde :

$$<t>::=\ \text{INTEGER}\mid\text{REAL}\mid\text{DOUBLE PRECISION}\mid\text{LOGICAL}\mid\text{COMPLEX}\mid\text{CHARACTER}$$

$$<nume>::=<identificator>$$



Scanned with OKEN Scanner

*p* sînt parametri formali și pot fi variabile neindexate, nume de variabile indexate, nume de procedură (în acest caz, în programul apelant se impune instrucțiunea EXTERNAL pentru numele de subprograme utilizate ca parametri).

Numele funcției trebuie să apară ca membru stîng al unei instrucțiuni de atribuire. În momentul atingerii unei instrucțiuni RETURN (sfîrșitul logic al subprogramului) această valoare a funcției este accesibilă programului apelant.

Un subprogram FUNCTION nu poate să conțină instrucțiunile SUBROUTINE, PROGRAM, o altă FUNCTION, BLOCK DATA. Apelarea se face utilizînd numele funcției (urmat de parametri actuali) ca operand în cadrul unor expresii. Ca parametri actuali se pot utiliza: variabile (indexate sau nu), nume de funcții externe sau predefinite, nume de variabile indexate, expresii, constante.

Observații. 1. Dacă tipul funcției este explicit definit, trebuie ca în programul apelant să se declare ca atare (dacă, de exemplu, se definește funcția INTEGER ADUNA (A,B) trebuie să apară instrucțiunea INTEGER ADUNA în programul apelant).

2. Dacă se utilizează o componentă de variabilă indexată ca parametru actual, aceasta echivalează cu a considera ca parametri actuali toate componentele ce urmează acesteia (dacă se consideră DIMENSION A (10,5) și se ia ca parametru actual componenta A (8,4), rezultă că se vor considera ca parametri componentele A (8,4), A (9,4), A (10,4), A (1,5), A (2,5), ..., A (10,5)).

3. La varianta actuală a compilatorului FELIX C-256 nu este operativ tipul CHARACTER și nici instrucțiunea PROGRAM utilizată în ANS FORTRAN (1976) și care marchează programele principale (are structura PROGRAM < identificator >).

4. Numele funcției nu se poate utiliza și ca nume de variabilă globală sau locală.

5. Parametrii formali nu pot apărea în instrucțiunile EQUIVALENCE, COMMON, DATA, INTRINSIC, PARAMETER (ultimele două neimplementate la FELIX C-256).

6. Sfîrșitul fizic este marcat de instrucțiunea END.

Subprogramele de tip SUBROUTINE se definesc astfel:

SUBROUTINE < *nume* > [(*p* [, *p*] ...)]

unde:

< *nume* > : : = < *identificator* >

*p* sînt parametri formali (sînt de tipul celor dați la FUNCTION), în plus, putînd apărea * care permite revenirea în programul apelant în altă parte decît la instrucțiunea imediată următoare lui CALL.

Apelul unui subprogram se face prin instrucțiunea CALL a cărei formă este

CALL < *nume* > [(*a* [, *a*] ...)]

unde *a* sînt parametri actuali (de tipurile specificate la FUCNTION, în plus putînd să apară *&e*, *e* fiind eticheta). Atingerea unui RETURN are



Scanned with OKEN Scanner

ca efect revenirea, în programul apelant, la instrucțiunea imediat următoare lui CALL.

Dacă se atinge un RETURN *i*, atunci se revine, în programul apelant, la eticheta specificată prin apel, corespunzătoare celei de a *i*-a steluță (de la stînga la dreapta) utilizată ca parametru formal.

Restricțiile lucrului cu SUBROUTINE sînt de aceeași natură ca cele de la FUNCTION.

Atît pentru subprogramele de tip FUNCTION cît și pentru cele de tip SUBROUTINE se poate utiliza instrucțiunea ENTRY :

ENTRY < *ni* > [(*p* [, *p*] ...)]

unde :

< *ni* > : : = < *identificator* >, *ni* fiind numele punctului de intrare în subprogram, altul decît cel normal dacă este vorba de SUBROUTINE și este nume de funcție dacă este vorba de FUNCTION ;

*p* sînt parametri formali.

4.1.7.3. *Subprograme* BLOCK DATA. Acestea sînt utilizate pentru inițializarea blocurilor de date comune etichetate. Instrucțiunea BLOCK DATA are structura :

BLOCK DATA < *identificator* >

În cadrul unei proceduri BLOCK DATA sînt permise numai instrucțiunile DIMENSION, DATA, COMMON, EQUIVALENCE, PARAMETER, declarațiile de tip și END care marchează sfîrșitul logic și fizic al procedurii.

4.1.8. **Fișiere.** Limbajul FORTRAN acceptă două categorii de fișiere : fișiere sistem și fișiere utilizator.

4.1.8.1. *Fișiere sistem.* Compilatorul FORTRAN FELIX C-256 inserează automat tabelele de descriere a două fișiere : fișierul de lucrări și fișierul de ieșire. Instrucțiunile standard corespunzătoare au forma

[< *e* >] READ < *el* > [,< *listă* >]
< *el* > FORMAT (< *listă de descriptori* >)

pentru fișierul de lucrări (de intrare) și

[< *e* >] PRINT < *el* > [,< *listă* >]
< *el* > FORMAT ('*c*',< *listă de descriptori* >)

pentru fișierul de ieșire.

Lista de descriptori se utilizează pentru a asigura conversia datelor de pe suport extern în memorie pentru READ și conversia inversă pentru



Scanned with OKEN Scanner

PRINT. La formatul pentru PRINT mai apare caracterul de control $c$ ($c=1$ dacă se dorește salt la pagină nouă, $c=0$ pentru scriere la două rînduri, $c=+$ dacă se scrie pe același rînd și se va scrie la un rînd pentru orice altă valoare a lui $c$, afară de 1, 0, +).

Limbajul FORTRAN admite mai mulți descriptori, dar în cele ce urmează ne vom referi la următoarii : *n*I*m* pentru a descrie *n* cîmpuri numerice a *m* caractere fiecare cu valori întregi ; *n*F*m*.*p* pentru a descrie *n* cîmpuri numerice a *m* caractere dintre care *p* constituie partea fracționară a numerelor reale ; *n*A*m* pentru a descrie *n* cîmpuri a cîte *m* caractere alfanumerice ; *n*L*m* pentru a descrie *n* cîmpuri a cîte *m* caractere pentru variabile de tip logic (dacă primul caracter din cîmp este T se va citi/scrie o valoare logică adevărată, iar în celelalte cazuri se va citi/scrie valoarea logică fals) ; *n*X pentru a neglija (insera) *n* spații (cînd $n=1$ se va scrie 1X).

Listele de descriptori sînt constituite din tipurile de descriptori amintite și, eventual, constante alfanumerice definite în unul din cele două moduri cunoscute.

Observații. 1. O instrucțiune READ, respectiv PRINT, se consideră executată fie la epuizarea listei cu atingerea parantezei finale a formatului, fie prin epuizarea listei și atingerea primului descriptor diferit de constantă alfanumerică.

2. Atingerea parantezei finale a unui FORMAT, fără să se fi epuizat lista, are ca efect trecerea la altă înregistrare cu reluarea formatului, de la perechea de paranteze situată cel mai aproape de paranteza finală, de atîtea ori pînă se epuizează lista.

Există și un al treilea fișier sistem, pentru care se folosește instrucțiunea

[< *e* >] PUNCH < *el* > [, < *listă* >]
< *el* > FORMAT (< *listă de descriptori* >)

similară lui PRINT, numai că este vorba de ieșire pe cartele.

**Exemple**

```
      LOGICAL A
      COMPLEX B
      READ 1,N,X,A,B,C
    1 FORMAT (I1,5X,F2.0,L1,2F3.1,A3)
      PRINT 2, N, X
    2 FORMAT ('0', I1, F5.1, L3, 2F5.1, 3X, A2, '*')
      READ 2
      N=N+N
      X=X+1.1
      PRINT 2,N,X,A,B,C
```

Dacă prima cartelă ce se citește conține în primele 18 coloane :

12345678T111222ABC

atunci : N=1 ; X=78. ; A=.TRUE. ; B=(11.,1, 22.2) ; C=ABCb. Efectul primei tipăriri : după ce se lasă un rînd liber, în primele șase coloane ale rîndului se vor scrie 1b78.0. Citirea a doua, fiind fără listă, se consideră încheiată după atingerea descriptorului A.



Scanned with OKEN Scanner

scriptorului F5.1. Dacă se citește o cartelă ce conține 1 în coloana 1, atunci formatul cu numărul 2 se păstrează, modificîndu-se zero, de pe post de caracter de control, în 1 (ceea ce s-a citit). Reutilizarea formatului 2, prin a doua instrucțiune PRINT, are ca efect tipărirea unui rînd pe pagină nouă (rolul lui READ 2 pentru o cartelă cu 1 în coloana 1 și + în coloana 26 a fost acela al modificării lui 0 în 1 și al trecerii lui + în locul steluței) cu următoarele caractere în primele 25 de coloane :

2b79.1bbTb11.1b22.2bbbAB+

4.1.8.2. *Fișiere utilizator.* Există trei tipuri : fișierele secvențiale, fișiere cu acces direct, fișiere pe bandă exploatate cu subprogramele BUFFER IN și BUFFER OUT.

Pentru fișierele sistem instrucțiunile au următoarele forme (la FELIX C-256) :

[< *e* >] READ (105, < *e*1 >, END=< *e*2 >) [< *listă* >]
< *e*1 > FORMAT (< *listă de descriptori* >)

dacă este vorba de fișierul de lucrări ;

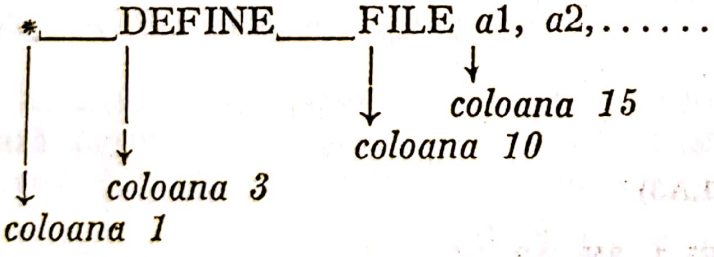
[< *e* >] WRITE (108, < *e*1 >) < *listă* >
< *e*1 > FORMAT ('*c*', < *listă de descriptori* >)

[< *e* >] WRITE (106, < *e*1 >) < *listă* >
< *e*1 > FORMAT (< *listă de descriptori* >)

dacă este vorba de ieșire la imprimantă, respectiv pe cartele.

Pentru alte fișiere, afară de cele menționate, utilizatorul are la dispoziție ordinele DEFINE FILE și USED FILE care au ca rol să furnizeze informațiile referitoare la suport, mod de organizare, mod de acces etc.

Structura pentru DEFINE FILE :




Primul spațiu ce apare după coloana 15 marchează sfîrșitul ordinului. Deci, la scrierea argumentelor nu este permisă utilizarea spațiului. Continuarea (dacă este necesară) se face scriind caracterul * în coloana 72 a liniei curente și în coloana 1 a liniei de continuare. Pentru linia de continuare argumentele se scriu din coloana 12 ; nu se acceptă despărțirea argumentelor.

O b s e r v a ț i e. Singurul loc unde pot apărea spații este la sfîrșitul unei linii care mai are continuare.



Scanned with OKEN Scanner

Argumentele au forma

$$*n=m1=m2=\ldots=mp$$

pentru fişierele sistem, cu $n=1,2,3$ după cum este vorba de fişierul de lucrări, fişierul la imprimantă, respectiv la perforatorul de cartele. Numerele $m1$, $m2$, ..., $mp$ sînt numerele ce se vor putea utiliza în scrierea instrucţiunilor. Dacă, de exemplu, se consideră

* DEFINE FILE *1=40=50,

înseamnă că numerele FORTRAN pentru cititorul de cartele sînt 105, 40, 50.

Pentru restul fişierelor argumentele au forma

$$an\,(p1,\ p2,\ \ldots,\ pr)=m1=m2=\ldots=mp$$

unde $a$ este o literă şi $n=1,2,0$, după cum fişierul este de intrare, de ieşire, respectiv de intrare/ieşire. Cu $p1$, $p2$ ... $pr$ s-au notat parametrii (tabelul 4.2), iar $m1$, $m2$, ... $mp$ au interpretarea dată.

*Tabelul 4.2*

**Parametrii pentru descrierea fişierelor în FORTRAN**

| Nr. crt. | Parametrul | Ce indică | Ce valori por lua : |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1. DVT : *tt* (DeVice Tape) | Tipul dispozitivului ; este parametru obligatoriu | *tt*<br>MT (Magnetic Tape)<br>RD (Removable Disc)<br>FD (Fixe Disc)<br>MD (Mass Disc)<br>CR (Card Reader)<br>CP (Card Punch)<br>PR (Printer) | |
| 2. RCF : *f* (ReCord Format) | Formatul articolelor ; este parametru opţional (implicit, tipul articolelor este nedefinit) | *f*<br>U (fişier secv., format nedefinit)<br>F (fişier secv., format fix)<br>FB (fişier secv., fix bloeat)<br>V (fişier secv., variabil)<br>VB (fişier secv., variabil blocat)<br>D (fişier cu acces direct)<br>B (fişier secv. utilizat cu BUFFER IN, BUFFER OUT) | |
| 3. RCS : *l* (ReCord Size) | Lungimea articolului în octeţi ; este obligatoriu pentru FB, VB, D | Pentru U, F, V, B, (din RCF) se ignoră RCS, considerîndu-se RCS : 256. | |



Scanned with OKEN Scanner

*Tabelul 4.2* (continuare)

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 4. | BFS : *n*<br>(BuFfer Size)<br>$n \leqslant 65535$ | Lungimea zonei tampon (buffer) în octeți | Este ignorat cînd RCF : B și se ia $n=256$ ($n$ este multiplu de 4 plus 4 sau 8, depinzînd de natura fișierului — bandă, disc) |
| 5. | BFN : *p*<br>(BuFfer Number) | Numărul de zone tampon | *p* poate fi 1 sau 2 (implicit se lucrează cu $p=2$) |
| 6. | MDO : *sss*<br>(MeDia Organisation) | Tipul de organizare<br>(numai pentru bandă) | *sss*<br>STD pentru organizare standard<br>NST pentru organizare nestandard<br>Implicit : MDO : STD |

Ordinul USED FILE are aceleași funcțiuni și sintaxă ca ale lui DEFINE FILE, numai că nu inserează tabelul de descriere a fișierului de lucrări și modulele de acces aferente, în cazul cînd acest fișier nu este utilizat de program.

Pentru detalii privind lucrul cu fișiere se pot consulta, de exemplu, lucrările [17, 56, 122, 123].

O b s e r v a ț i e. Pentru a oferi programelor o mai mare independență în ceea ce privește formatele de intrare/ieșire (capitol la care limbajul FORTRAN este deficitar) se utilizează instrucțiunile :

[< *e* >] READ (105,*v*) < *listă* >
[< *e* >] WRITE (108,*v*) < *listă* >
[< *e* >] WRITE (106,*v*) < *listă* >

unde cu *v* s-a notat numele unei variabile indexate în care, înainte de utilizarea acestor instrucțiuni se introduce formatul de citire/scriere (în variabila indexată se introduce totul, începînd cu paranteza ce se scrie după cuvîntul FORMAT în variantă normală).

**Exemple**

```
      DIMENSION F (20)
      ...
      READ (105,1)F                    citirea formatului dorit
      ...
    1 FORMAT (20A4)
      ...
      READ (105,F)A,K,B,J,L            citirea conf. formatului citit
      ...
      READ (105,1)F                    citirea formatului de scrier
      ...
      WRITE (108,F)X,A,J               scrierea cu formatul citit
      ...
```



Scanned with OKEN Scanner

## 4.2. LIMBAJUL ALGOL

Limbajul ALGOL este primul limbaj de programare elaborat de un colectiv internațional, reunit la lucru în 1957. Un prim rezultat al efortului colectivului a fost limbajul IAL (International Algorithmic Language), care a stat apoi la baza elaborării limbajului ALGOL 58 (ALGOrithmic Language) în 1958. După încă doi ani de cercetări și căutări se ajunge la varianta ALGOL 60, prezentată sub forma unui raport („Report on the Algorithmic Language ALGOL 60").

Mergîndu-se pe calea perfecționării limbajului, se realizează un raport îmbunătățit („Revised Report on the Algorithmic Language ALGOL 60"). Prin îmbunătățirile aduse și formei ALGOL 60 revizuit, s-a creat varianta ALGOL 68, fără a fi însă răspîndit în rîndul utilizatorilor.

Deși este unul dintre limbajele de programare complexe, limbajul ALGOL (în varianta ALGOL 60 revizuit, care este standardizată) are o răspîndire mare. Aceasta se datorește apropierii, ca mod de scriere, de scrierea matematică uzuală și rigurozității definirii care permite cunoașterea precisă a posibilităților sale.

Limbajul ALGOL 60 este cunoscut sub trei forme :

— *limbajul de referință*, forma în care apare definit în raportul revizuit ;

— *limbajul de lucru*, forma în care limbajul este acceptat de un sistem de calcul ;

— *limbajul de publicare*, formă mai puțin restrictivă care ușurează publicarea programelor și schimbul de informații între specialiști.

Limbajul de referință nu are în vedere intrările/ieșirile, ceea ce impune ca atunci cînd se concepe compilatorul ALGOL pentru un anumit tip de calculator, să se definească un mod de realizare a introducerii și extragerii informațiilor. În plus, nu se pot utiliza toate posibilitățile limbajului de referință care conțin în sine și o serie de idealizări ce trebuie înlăturate prin soluții care diferă de la un calculator la altul.

În cele ce urmează se va prezenta limbajul de referință ALGOL 60, varianta revizuită.

4.2.1. **Simbolurile de bază** ale limbajului ALGOL se împart în patru categorii (*litere, cifre, valori logice, delimitatori*) :

$$< \textit{simbol de bază} > ::= < \textit{literă} > \mid < \textit{cifră} > \mid < \textit{valoare logică} > \mid < \textit{delimitator} >$$

1) *Litere.* Există 52 de litere, literele minuscule și majuscule ale alfabetului englez :

$$< \textit{literă} > ::= a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z$$



Scanned with OKEN Scanner

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
P | R | Q | S | T | U | V | W | X | Y | Z

Pentru diferite forme de limbaje de lucru ALGOL se utilizează cel mult aceste 52 de litere. De cele mai multe ori se utilizează numai cele 26 de litere majuscule.

2) *Cifre.* Sînt utilizate cifrele sistemului de numeraţie zecimal :

< *cifră* > : : =0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

3) *Valori logice.* În limbajul ALGOL se lucrează cu propoziţii din logica cu două valori. Aşadar, există

< *valoare logică* > : : = **true** | **false**

Se remarcă faptul că unele cuvinte englezeşti au un rol aparte. Acestea se vor sublinia (în unele publicaţii ele sînt culese cu litere aldine), fiind considerate ca simboluri unice, indivizibile, aşa cum sînt literele, cifrele.

Cuvintele englezeşti utilizate ca simboluri de bază sînt următoarele :

| | | | |
|---|---|---|---|
| **true** | **for** | **begin** | **array** |
| **false** | **do** | **end** | **switch** |
| **go to** | **step** | **own** | **procedure** |
| **if** | **until** | **Boolean** | **string** |
| **then** | **while** | **integer** | **label** |
| **else** | **comment** | **real** | **value** |

4) *Delimitatorii* se împart în cinci grupe :

< *delimitator* > : : = < *operator* > | < *separator* > | < *paranteză* > | < *declarator* > | < *specificator* >

a) < *operator* > : : = < *operator aritmetic* > | < *operator relaţional* > | < *operator logic* > | *operator secvenţial* >

< *operator aritmetic* > : : = + | − | × | / | ÷ | ↑

< *operator relaţional* > : : = < | ⩽ | = | ⩾ | >

< *operator logic* > : : = ⊃ | ≡ | ∨ | ∧ | ¬

< *operator secvenţial* > : : = **go to** | **if** | **then** | **else** | **for** | **do**

Operatorii aritmetici corespund, respectiv, operaţiilor adunare, scădere, înmulţire, împărţire, împărţire întreagă, ridicare la putere, iar operatorii logici corespund, respectiv, implicaţiei, echivalenţei, disjuncţiei, conjuncţiei, negaţiei. Cei şase operatori secvenţiali se vor utiliza în vederea realizării de succesiuni de instrucţiuni dorite de programator.

b) < *separator* > : : =, | . | $_{10}$ | : | ; | : = | — | **step** | **until** | **while** | **comment**

c) < *paranteză* > : : =( | ) | [ | ] | ‘ | ’ | **begin** | **end**



Scanned with OKEN Scanner

Se remarcă aici că există perechea de simboluri **begin** şi **end** care sînt considerate ca paranteze, fiind utilizate la constituirea de blocuri atunci cînd se scriu programe.

d) < *declarator* > : : = **own | Boolean | integer | real | arrany | switch | procedure**

Aceştia se vor utiliza la constituirea declaraţiilor.

e) < *specificator* > : : = **string | label | value**

Specificatorii servesc la scrierea specificaţiilor în proceduri.

4.2.2. **Structura programelor.** Programele ALGOL sînt compuse din instrucţiuni, instrucţiuni compuse, blocuri. Dacă se notează cu I, D, E respectiv instrucţiuni, declaraţii, etichete, atunci instrucţiunile compuse au structura

[*E* : [*E* : ] . . . ] **begin** *I*;*I* ; . . . ; *I*;*I* **end**

iar blocurile au structura

[*E* : [*E* :] . . . ] **begin** *D*;*D* ; . . . ; *D*;*I*;*I* ; . . . ;*I* **end**

Orice program ALGOL este sau instrucţiune compusă sau bloc :

< *program* > : : = < *instrucţiune compusă* > | < *bloc* >

Structura unui program ALGOL este dată în fig. 4.3.

**Exemple**

*Instrucţiuni*

```
x:=m+n/k+x↑2
goto sfîrsit
ÎNCEPUT : RELUARE : A [i,j] : =TAU
```

*Instrucţiune compusă*

```
X : Y : begin s : =0 ; for k : =1 step 2 until n do s=s+x↑k ; if
        s=100.5 then go to STOP else if s=200 go to R ; el : r=s+s
        end
```

*Bloc*

```
B : begin integer a, b ; real u ; for a : =1 step 1 until n do for b : =a+1
    step 1 until m do begin u : =A [a, b] ;
                            A [a,b] : =A [b, a] ;
                            A [b,a] : =u
                            end pentru a şi b
    end B
```



Scanned with OKEN Scanner

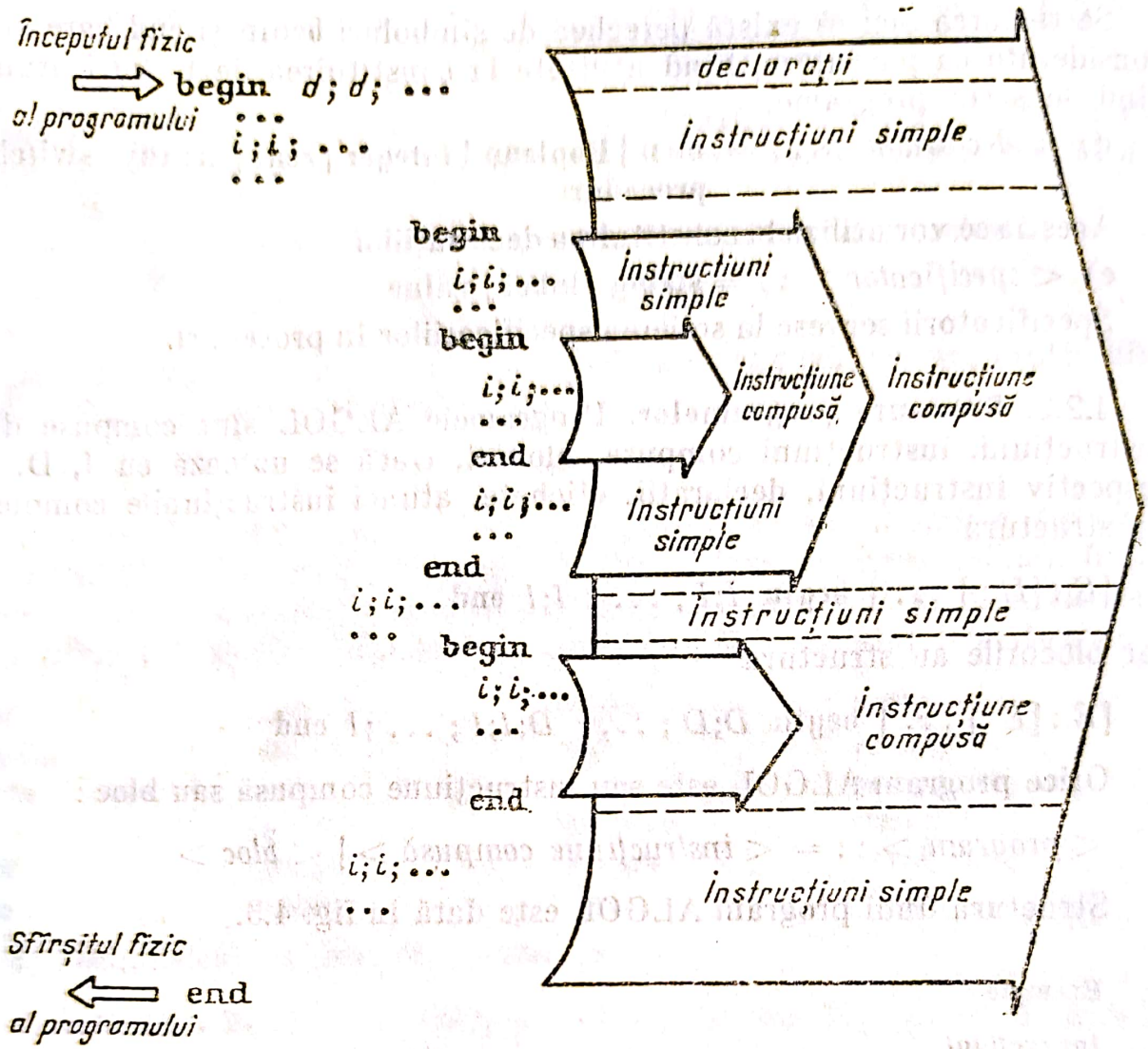



Fig 4.3. Structura unui program ALGOL.

4.2.3. **Identificatorii** sînt succesiuni de litere sau cifre, primul dintre caractere fiind literă. Deci :

$$< identificator > : := < literă > \mid < identificator > < literă > \mid < identificator > < cifră >$$

Identificatorii sînt utilizaţi pentru a desemna variabile simple, tablouri, etichete, proceduri.

**Exemple**

| | | |
|---|---|---|
| Indice | AUXILIARĂ | RADICALINDICE2 |
| A1 | T1234567894 | Sistem |
| XY47 | tabel | Valoare |
| TABEL1 | ZETA | INTREG |
| Valoare | rezultat | FuNcToR |

4.2.4. **Numere.** Limbajul ALGOL operează cu următoarele tipuri de constante numerice (numere): întreg fără semn, întreg, fracție zecimală, exponent, număr zecimal, număr fără semn. Acestea se definesc după cum urmează:

< *întreg fără semn* > : : =< *cifră* > | < *întreg fără semn* > < *cifră* >

< *întreg* > : : =< *întreg fără semn* > | {±} < *întreg fără semn* >

< *fracție zecimală* > : : =. < *întreg fără semn* >

< *exponent* > : : $=_{10}$ < *întreg* >

< *număr zecimal* > : : =< *întreg fără semn* > | < *fracție zecimală* > | < *întreg fără semn* > < *fracție zecimală* >

< *număr fără semn* > : : =< *număr zecimal* > | < *exponent* > | < *număr zecimal* > < *exponent* >

< *număr* > : : =< *număr fără semn* > | + < *număr fără semn* > | − < *număr fără semn* >

**Exemple**

| *Scrierea în* ALGOL | *Tipul constantei* | *Scrierea obișnuită* |
|---|---|---|
| 1937 | întreg fără semn | 1937 |
| −175 | întreg | −175 |
| .8 | fracție zecimală | 0,8 |
| $_{10}-4$ | exponent | $_{10}-4$ |
| −10.56 | număr | −10,56 |
| $-_{10}+3$ | număr | $-10^3$ |
| $10.5_{10}-03$ | număr fără semn | $10{,}5 \cdot 10^{-3}$ |

4.2.5. **Șirurile** admise de limbajul ALGOL se definesc după cum se arată în continuare.

< *șir propriu* > : : =< *succesiune de simboluri de bază exceptînd 'sau'* > | < *loc gol* >

< *șir deschis* > : : =< *șir propriu* > | '< *șir deschis* >' | < *șir deschis* > < *șir deschis* >

< *șir* > : : ='< *șir deschis* >'



Scanned with OKEN Scanner

**Exemple**

'Ex./+'XTAU=/'ab+aba'
'UN alt eXeMpLu/–'+++/–––"ALGOL FORTRAN"

4.2.6. **Expresiile** se definesc astfel:

< *expresie* > : : =< *expresie aritmetică* > | < *expresie indicatoare* > < *expresie booleană* >

Expresiile aritmetice sînt reguli de calcul numeric, expresiile booleene sînt reguli de calcul logic, iar expresiile indicatoare au drept rezultat un marcaj (o etichetă).

4.2.7. **Variabile**

< *identificator de variabilă* > : : =< *identificator* >

< *variabilă simplă* > : : =< *identificator de variabilă* >

< *expresie indicială* > : : =< *expresie aritmetică* >

< *listă de indici* > : : =< *expresie indicială* > | < *listă de indici* >, < *expresie indicială* >

< *identificator de tablou* > : : =< *identificator* >

< *variabilă indexată* > : : =< *identificator de tablou* > [< *listă de indici* >]

< *variabilă* > : : =< *variabilă simplă* > | < *variabilă indexată* >

**Exemple**

| | | |
|---|---|---|
| A [4,19] | TAU [n, kp/2, ia] | Beta 4 [nm+nm, nm/n] |
| XYZETA | alfa | epsilonescu [1, n, q, –1, 0] |

Tipurile de variabile sînt:

< *tip de variabilă* > : :=**integer** | **real** | **Boolean**

Indicii variabilelor în ALGOL, spre deosebire de alte limbaje (de exemplu, limbajul FORTRAN) pot căpăta și valori mai mici ca zero sau chiar zero. Așadar, indicii sînt numere întregi care se încadrează în limitele date prin declararea variabilelor indexate (tablourilor).

4.2.8. **Indicatorii funcționali** pun în evidență funcțiile utilizate în scrierea programelor.

< *identificator de procedură* > : : =< *identificator* >

< *parametru actual* > | | =< *șir* > | < *expresie* > | < *identificator de tablou* > | < *identificator de conector* > | < *identificator de procedură*>



Scanned with OKEN Scanner

< *șir de litere* > : : = < *literă* > | < *șir de litere* > < *literă* >

< *delimitatori de parametri* > : : = , | ) < *șir de litere* > : (

< *listă de parametri actuali* > : : = < *parametru actual* > |

< *listă parametri actuali* > < *delimitator de parametri* > < *parametru actual* >

< *partea parametrilor actuali* > : : = < *loc gol* > | (< *listă de parametri actuali* >)

< *indicator funcțional* > : : = < *identificator de procedură* > < *partea parametrilor actuali* >

**Exemple**

| | |
|---|---|
| sin (x+pi/2) | F (x,y,z) |
| COS (x ↑ 2−y ↑ 2) | det (A) |
| Test ('x⩽y') | CALCUL |
| Spațiu (S) Viteza : (V) Timp : (t) | |

La ultimul exemplu se văd separatorii ) Viteza : ( și ) Timp : (, numele funcției fiind Spațiu (cei doi separatori sînt sugestivi pentru parametrii V și t.

Indicatorii funcționali definesc o valoare numerică sau logică care rezultă prin aplicarea unor reguli de calcul (definite printr-o declarație de procedură) asupra parametrilor actuali.

În ALGOL există nouă funcții standard (pot fi utilizate fără a fi nevoie de declararea lor) :

| | |
|---|---|
| abs (E) | valoarea absolută a expresiei E ; |
| sign (E) | semnul unor expresii ; funcția va lua valoarea 1 dacă E>0, 0 dacă E=0 și −1 dacă E<0 |
| sqrt (E) | calculează radicalul de ordinul doi din E ; |
| sin (E) | calculează sinusul lui E ; |
| cos (E) | calculează cosinusul lui E ; |
| arctan (E) | calculează arcul a cărui tangentă este E ; |
| ln (E) | calculează logaritmul natural din E ; |
| exp (E) | calculează $e^E$ ; |
| entier (E) | ia partea întreagă a lui E. |

Denumirile funcțiilor standard sînt rezervate, fiind interzisă utilizarea lor în alte scopuri.

4.2.9. **Expresii aritmetice.** Se va da mai întîi definiția expresiei aritmetice simple și apoi a expresiei aritmetice.

< *operator aditiv* > : : = + | −

< *operator multiplicativ* > : : = / | ÷ | ×



Scanned with OKEN Scanner

< *element primar* > : : = < *număr fără semn* > | < *variabilă* > |
< *indicator funcțional* > |(< *expresie aritmetică* >)

< *factor* > : : =< *element primar* > | < *factor* > ↑ < *element primar* >

< *termen* > : : =< *factor* > | < *termen* > < *operator multiplicativ* > < *factor* >

< *expresie aritmetică simplă* > : : =< *termen* > | < *operator aditiv* > < *termen* > | < *expresie aritmetică simplă* > < *operator aditiv* > < *termen* >

Pentru a ajunge la definiția expresiei aritmetice se mai definește clauza *if* :

< *clauza if* > : : =**if** < *expresie booleană* > **then**

avînd :

< *expresie aritmetică* > : : =< *expresie aritmetică simplă* > | < *clauza if* > < *expresie aritmetică simplă* > **else** < *expresie aritmetică* >

**Exemple**

| | |
|---|---|
| 1937 | alfa |
| X | sxcos (a−b) |
| sin (A) | $7_{10}$−4/alfa/sqrt (t) |
| (A−b) ↑ 2 | A [K+1,3] ↑ (x−3/sqrt (a)) |
| A [i,−1,x] | x [i,j]/y [k,j] |
| UV+Z−alfa ↑ b | **if** B=0 **then** A/B **else** A ↑ 2 |
| **if** a **then** k−1 **else** k | **if** x<0 **then** A+B **else if** x=0 **then** A−B **else** x |

Observații. 1. Dacă *i* este variabilă întreagă și reală, atunci :

$$z=a\uparrow i=\begin{cases} a\times a\times\ldots\times a \text{ (de } i \text{ ori } a), \text{ dacă } i>0, z \text{ având același tip cu } a; \\ 1, \text{ dacă } a=0 \text{ și } i=0, z \text{ având același tip cu } a; \\ 1/(a\times a\times\ldots\times a), \text{ dacă } a\neq 0 \text{ și } i<0, z \text{ fiind de tip real } (z \text{ eset nedefinit dacă } a=0); \end{cases}$$

$$y=a\uparrow r=\begin{cases} e\times p\,(r\times ln\,(a)), \text{ dacă } a>0, y \text{ fiind real}; \\ 0, \text{ dacă } a=0 \text{ și } r>0 \text{ (cînd } r<0, y \text{ nedefinit)}; \\ \text{nedefinit, dacă } a<0. \end{cases}$$

2. Evaluarea expresiilor aritmetice se face ca în FORTRAN.

4.2.10. *Expresii booleene.* Sintaxa expresiilor booleene este următoarea :

< *operator relațional* > : : =< | ⩽ | = | > | ≠ | ⩾

< *relație* > : : =< *expresie aritmetică simplă* > < *operator relațional* > < *expresie aritmetică simplă* >

< *element boolean primar* > : : =< *valoare logică* > | < *variabilă* >|



Scanned with OKEN Scanner

< *indicativ funcțional* > | < *relație* > | (< *expresie booleană* >)

< *element boolean secundar* > : : = < *element boolean primar* > | ¬ < *element boolean primar* >

< *factor boolean* > : : = < *element boolean secundar* > | < *factor boolean* > ∧ < *element boolean secundar* >

< *termen boolean* > : : = < *factor boolean* > | < *termen boolean* > ∨ < *factor boolean* >

< *implicație* > : : = < *termen boolean* > | < *implicație* > ⊃ < *termen boolean* >

< *expresie booleană simplă* > : : = < *implicație* > | < *expresie booleană simplă* > ≡ < *implicație* >

< *expresie booleană* > : : = < *expresie booleană simplă* > | < *clauza if* > < *expresie booleană simplă* > **else** < *expresie booleană* >

**Exemple**

val=−56.5
A>B∨x≠y
s+t/2≠−10∧(a−b)↑2<45
a≡¬alfa∧x∨t∧xy⊃aux
**if** i<0 **then** x≥y **else** x=0
**if if if** x **then** y **else** z **then** u **else** v **then** w **else** t≥s

*Evaluarea expresiilor booleene* se face ca la expresiile logice în FORTRAN, ținînd seama că funcțiile ¬, ∧, ∨, ⊃, ≡ sînt definite astfel:

| *a* | *b* | ¬*a* | *a*∧*b* | *a*∨*b* | *a*⊃*b* | *a*≡*b* |
|---|---|---|---|---|---|---|
| **false** | **false** | **true** | **false** | **false** | **true** | **true** |
| **false** | **true** | **true** | **false** | **true** | **true** | **false** |
| **true** | **false** | **false** | **false** | **true** | **false** | **false** |
| **true** | **true** | **false** | **true** | **true** | **true** | **true** |

**Exemplu de evaluare.** Să se evalueze expresia:
**if if** u∧w **then** m+n≤8 **else**¬u **then** t≥0 **else** b≡a∧¬b∧(x≥y∨¬x≠y)∧a știind că: u=w=**true**, m=4, n=3, t=1, a=**true**, b=**false**, x=y=2.

**if if** u∧w **then** m+n≤8 **else**¬u **then** t≥0 **else** b≡a∧¬b∧(x≥y∨¬x≠y)∧a

**true** **true**

**true** (deoarece u∧w=**true** și m+n=7≤8) **true**

**true** deoarece t=10 și rezultatul anterior este **true** **true**

**true**

4.2.11. **Expresiile indicatoare** sînt reguli de determinare a etichetelor unor instrucţiuni. Evaluarea lor este analoagă cu cea a expresiilor aritmetice. Sintactic, sînt definite astfel :

< *label* > : : =< *identificator* > | < *număr fără semn* >

< *identificator de comutator* > : : =< *identificator* >

< *identificator de comutator* > : : =< *identificator de comutator* > | [< *expresie indicială* >]

< *expresie indicatoare simplă* > : : =< *etichetă* > | < *indicator de comutator* > | (< *expresie indicatoare simplă* >)

< *expresie indicatoare* > : : =< *expresie indicatoare simplă* > | < *clauză if* > < *expresie indicatoare simplă* > **else** < *expresie indicatoare* >

**Exemple**

xy1
1976
Eticheta [K]
I [if x≠0 then K else J]
if x=0 then 13 else g [if u then 4 else i ]

4.2.12. **Instrucţiuni de atribuire.** Sintaxa instrucţiunilor de atribuire este următoarea :

< *parte stîngă* > : : =< *variabilă* > : = | < *identificator de procedură* > : =< *listă de părţi stîngi* > : : =< *parte stîngă* > | < *listă de părţi stîngi* > < *parte stîngă* >

< *instrucţiune de atribuire* > : : =< *listă de părţi stîngi* > < *expresie aritmetică* > | < *listă de părţi stîngi* > < *expresie booleană* >

**Exemple**

*Părţi stîngi*

a : =
mat [I, j, k] : =

*Listă de părţi stîngi*

x : =y : =z : =
A [I, J] : =B [I, J] : =i : =

*Instrucţiuni de atribuire*

x : =4.75
A [I, J, K] : =B>C∨L
Alfa [i, j+k] : =Beta [j+k, l−2] : =A≥B∧C≠D
L : =G : =J : =if a then b else true



Scanned with OKEN Scanner

După cum rezultă, instrucțiunile de atribuire servesc la calcularea unor valori de expresii aritmetice, respectiv logice, în scopul păstrării acestora în una sau mai multe variabile aritmetice, respectiv logice.

Toate variabilele prezente într-o listă de părți stîngi au același tip cu al expresiei ce intervine în definirea instrucțiunii.

4.2.13. **Declarațiile** folosesc la definirea tipurilor variabilelor ce intervin într-un bloc. Mai exact, acestea furnizează translatorului informațiile necesare utilizării corecte a valorilor variabilelor.

Cele patru tipuri de declarații sînt :

< *declarație* > : : =< *declarație de tip* > | < *declarație de tablou* > | < *declarație de conector* > | < *declarație de procedură* >

Aici se vor trata primele trei tipuri de declarații, urmînd ca declarația de procedură să se prezinte cînd se va vorbi despre proceduri.

1) *Declarația de tip* servește la precizarea tipurilor variabilelor simple.

< *listă de tipuri* > : : =< *variabilă simplă* > | < *variabilă simplă* >, < *listă de tipuri* >

< *tip* > : : = **real** | **integer** | **Boolean**

< *tip local sau remanent* > : : = < *tip* > | **own** < *tip* >

< *declarație de tip* > : : =< *tip local sau remanent* > < *listă de tipuri* >

**Exemple**

```
integer alfa, x, y, i, j, k
own real i, j, k
own integer a
Boolean unu, doi
```

2) *Declarația de taublou* se utilizează pentru a pune în evidență variabilele indexate (tablourile) cu care se lucrează, dînd informații privind :

– tipul elementelor tabloului ;
– numărul de indici ;
– limitele (inferioară și superioară) pentru indici.

< *margine inferioară* > : : =< *expresie aritmetică* >

< *margine superioară* > : : =< *expresie aritmetică* >

< *pereche de margini* > : : =< *margine inferioară* > : < *margine superioară* >



Scanned with OKEN Scanner

< *listă de perechi de margini* > : : =< *pereche de margini* > |
< *listă de perechi de margini* > < *pereche de margini* >

< *segment de tablou* > : : =< *identificator de tablou* > [< *listă de perechi de margini* >] | < *identificator de tablou* >, < *segment de tablou* >

< *listă de tablou* > : : =< *segment de tablou* > | < *listă de tablou* >, < *segment de tablou* >

< *declarație de tablou* > : : = **array** < *listă de tablou* > | < *tip local sau remanent* > **array** < *listă de tablou* >

**Exemple**

**integer array** i [0 : 4, −4 : 10], j, k [−6 : 5]
**real array** x [0 : **if** i=10 **then** 10 **else** 100]
**own Boolean array** A [**if** ×0 **then** 1 **else** 1 : 4]

Observații. 1. Tipul implicit este **real** (exemplul al doilea se putea scrie fără real).

2. În fața oricăruia dintre tipurile **integer, real, Boolean** se poate pune simbolul **own.**

3. Din exemplul 1 rezultă că j și k sînt variabile indexate cu un indice (vectori), de cîte 12 componente (j (−6), j (−5), . . ., j (−1), j (0), j (1), . . ., j (5), k (−6), k (−5), k (−4), . . ., k (−1), k (0), k (1), . . ., k (5), toate de tip întreg.

3) *Declarația de conector* are următoarea sintaxă :

< *lista conectorului* > : : =< *expresie indicatoare* > | < *lista conectorului* > , < *expresie indicatoare* >

< *declarație de conector* > : : = **switch** < *identificator de conector* > : =< *lista conectorului* >

**Exemple**

**switch** S : =et1, et2, et3,
**switch** Contor : = L1, **if** a=b **then** X **else** Y

În al doilea exemplu conectorul se numește Contor. Aceasta se va utiliza sub forma de indicator de conector :

Contor [< *expresie indicială* >]
calculul valorii sale decurgînd după cum urmează :

— se calculează valoarea expresiei indiciale ;

— dacă valoarea expresiei indiciale este i (cuprins între 1 și numărul de elemente ale listei conectorului), se selectează elementul i al listei menționate în declarația conectorului ;

— se calculează valoarea acestei expresii indicatoare.

Dacă se ia, de exemplu :

**switch** G :=L1, L2, **if** x≠y **then** A1 **else** A2

atunci indicatorul G (2×n−1), pentru n=2, x=2, y=3 se calculează astfel :
— valoarea expresiei indiciale este 3 (2×n−1=3) ;
— al treilea element în listă este : **if** x≠y **then** A1 **else** A2 şi conduce la valoarea A1 deoarece x≠y (2≠3).
Aşadar G (2×n−1)=A1.

4.2.14. **Instrucţiunea de transfer** are sintaxa :
< *instrucţiune de transfer* > : : = **go to** < *expresie indicatoare* >

**Exemple**

**go to** A
**go to** Alege [K+1]
**go to** Select [**if** x=y **then** m **else** n]
**go to if** L=**true then** contor [n−1] **else** E1

**Problemă.** Să se calculeze :

$$SPOZ=\sum_{\substack{i=1\\x_i>0}}^{50} x_i\ ;\quad SNEG=\sum_{\substack{i=1\\x_i<0}}^{50} x_i\ ;\quad STOT=SPOZ+SNEG$$

*Rezolvare.* Se presupune că cele 50 de componente ale vectorului se citesc cîte una, pentru citire utilizînd, de exemplu, instrucţiunea x [i] :=read
Pentru tipărirea rezultatelor se va utiliza

print (SPOZ, SNEG, STOT),

ceea ce se va însemna scrierea celor trei valori. În aceste condiţii programul este următorul :

```
begin integer i ; real SPOZ, SNEG, STOT ;
      array x [1 : 50] ; i :=0 ;
Citire : i : = i+1 ; x [i] : =read ;
go to if i<50 then Citire else Iniţializare ;
Iniţializare : SPOZ : =SNEG : =STOT : =0. ; i : =0 ;
Iar : i : =i+1 ; go to if x [i]<0 then NEG else POZ ;
NEG : SNEG : =SNEG+x [i] ; go to TEST ;
POZ : SPOZ : =SPOZ+x [i] ;
TEST : go to if i=50 then STOP esle Iar ;
STOP : STOT : =SNEG+SPOZ ;
      print (SPOZ, SNEG, STOT)
end
```

4.2.15. **Instrucţiunea condiţională** permite o utilizare mai flexibilă a instrucţiunilor unui program, atunci cînd utilizarea instrucţiunilor de transfer (go to) ar fi greoaie.
Sintaxa :

< *clauza if* : : = **if** < *expresie booleană* > **then**

< *instrucţiune necondiţională* > : : =< *instrucţiune de bază* > |
< *instrucţiune compusă* > | < *bloc* >



Scanned with OKEN Scanner

< *instrucțiune if* > : : = < *clauza if* > < **instrucțiune necondițională** >

< *instrucțiune condițională* > : : = < *instrucțiune if* > | < *instrucțiune if* > **else** < *instrucțiune* > | < *clauza if* > < **instrucțiune de ciclare** > | < *etichetă* > : < *instrucțiune condițională* >

**Exemple**

```
if a=0 then x:=x+y [i]
if a=b then x=x+1 else go to A
if a<0∧b<0 then eticheta: begin if t then x=0 else x=1 end else if w then x=2
                else go to terminare
```

4.2.16. **Instrucțiunea de ciclare.** Sintaxa instrucțiunii este următoarea :

< *element al listei ciclului* > : : = < *expresie aritmetică* > | < *expresie aritmetică* > **step** < *expresie aritmetică* > **until** < *expresie aritmetică* > | < *expresie Booleană* >
**while** < *expresie aritmetică* >

< *lista ciclului* > : : = < *element al listei ciclului* > | < *lista ciclului* > , < *element al listei ciclului* >

< *instrucțiune de ciclare* > : : = **for** < *variabilă* > : = < *lista ciclului* > **do** < *instrucțiune* >

**Exemple**

```
for k:=1 step 1 until m do X[k]:=A[k]
         lista ciclului      instrucțiunea

for k:=4, 2 step −1 until 0, −10, a do V:=V+K↑2
          în loc de: 2, 1, 0

for i:=A+B, C, 2 step 4 until MAX while i>k do X[i]:=0
          în loc de: 2, 6, 10,..., MAX
```

Observație. În cazul exemplului al doilea se calculează $4^2+2^2+1^2+0^2+4-(-10)^2+a^2$, iar în exemplul al treilea, dintre componentele X [A+B], X [C], X [2], X [6], . . ., X [MAX], se anulează numai acelea de indice i > K.

4.2.17. **Proceduri.** Folosirea unei proceduri presupune :

– declarația de procedură cu ajutorul căreia se definește procesul de calcul ;

– apelul procedurii.

Limbajul ALGOL admite două tipuri de proceduri : proceduri funcție (ce furnizează un singur rezultat) și proceduri instrucțiune (ce furnizează mai multe rezultate).

1) *Instrucțiunea de apel procedură:*

< *parametru actual* ▷ : : = ◁ *șir* > | < *expresie* > | < *nume de tablou* ▷ | ◁ *identificator de comutator* > | < *identificator de procedură* >

< *șir de litere* > : : = < *literă*> | <*șir de litere*> <*literă*>

◁ *delimitator de parametru* > : : =, | ) < *șir de litere* > : (< *listă de parametri actuali* > : : = < *parametru actual* > | < *listă de parametri actuali* > < *delimitator de parametru* > < *parametru actual* >

< *partea parametrilor actuali* > : : = < *loc gol* > | (< *listă de parametri actuali* >)

< *instrucțiune de procedură* > : : = < *identificator de procedură* > < *partea parametrilor actuali* >

2) *Declarația de procedură:*

< *parametru formal* > : : = < *identificator* >

◁ *listă de parametri formali* > : : = < *parametru formal* > | < *listă de parametri formali* > < *delimitatori de parametri* > < *parametru formal* >

◁ *partea parametrilor formali* > : : = < *loc gol* > | (< *listă de parametri formali* >)

◁ *listă de identificatori* > : : = < *identificator* > | < *listă de identificatori* > , < *identificator* >

◁ *parte de valoare* ▷ : : = **value** < *listă de identificatori* > ; | < *loc gol* >

< *specificator* > : : = **string** | < *tip* > | **array** | < *tip* > **array** | < **label** > | **switch** | **procedure** | < *tip* > **procedure**

< *parte de specificație* > : : = < *loc gol* > | < *specificator* > < *listă de specificatori* > ; | < *parte de specificație* > < *specificator* > < *listă de identificatori* > ;

< *titlul procedurii* > : : = < *identificator de procedură* > < *partea parametrilor formali* > ; < *partea de valoare* > < *partea de specificație* >

< *corpul procedurii* > : : = < *instrucțiune* > | < *cod* >

< *declarația de procedură* > : : = **procedure** < *titlul procedurii* >

< *corpul procedurii* > | < *tip* > **procedure** < *titlul procedurii* > < *corpul procedurii* >



Scanned with OKEN Scanner

Exemple. *Declarații de proceduri funcție*

```
real procedure sh (x) ; value x ; real x ;
    begin real aux ; aux : = exp (x) ;
                     sh : = (aux −1/aux)/2
    end ;
real procedure arcsin (x) ; value x ; real x ;
                arcsin : = arctan (x/sqrt (1−x ↑ 2)) ;
```

*Declarații de proceduri instrucțiune*

```
procedure ABSOLMAX (A) dimensiunea : (n, m) rez : (y) ind : (i, k) ;
          comment Calculul lui y (maximul în valoare absolută, din A și i, k indicii
acestei valori y) ;
          array A ; integer n, m, i, k ; real y ;
          begin integer p, q ; y : =0 ;
          for p : =1 step 1 until n do for q : =1 step 1 until m do if abs (A [p,
              q])>y then begin y : = abs (A [p, q]) ;
                               i : =p ; k : =q
                         end ;
          end ;
procedure exemplu (X) linii : (m) coloane : (n) ;
    value m, n ; array X ; integer m, n ;
    begin integer i, j ;
        for i : =1 step 2 until m do
        for j : =1 step 1 until i−1 do
            X [i, j] : =X [j, i]
    end ;
```

Exemple. *Apelare a procedurilor declarate*

```
w : = sh (a)
rez : = sh (a+b)−arcsin (b)
exemplu (A, linia, coloana)
exemplu (A, if m>n then i else j, k)
```

Observație. Corespondența parametri actuali — parametri formali se face în două moduri : prin valori și prin nume.

Partea de specificație din declarația de procedură conține obligatoriu specificații legate de parametrii chemați prin valoare (dacă aceștia există).

Procedurile ALGOL permit apelul recursiv. De exemplu, procedura MED3 de calculare a mediei aritmetice a trei numere :

```
real procedure MED3 (x, y, z) ;
    value x, y, z ; real x, y, z ;
MED3 : =(x+y+z)/3 :
```

permite o utilizare de forma :

```
S : = MED3 (MED3 (a, b, c), MED3 (a+b, u, 3), 9)
```

4.2.18. **Domeniul cantităților.** Prin cantități se desemnează următoarele noțiuni ALGOL : variabile simple, tablouri, etichete, conectori, proceduri. Domeniul unei cantități este totalitatea instrucțiunilor și expresiilor în care este valabilă declarația identificatorului asociat acestei cantități.



Scanned with OKEN Scanner

În general, o cantitate declarată într-un bloc A față de un bloc B se poate afla în una din situațiile :

– globală față de B dacă blocul B este inclus în A și aceasta nu a fost redeclarată în B ;

– locală față de B dacă s-a redeclarat în B ;

– nedefinită față de B dacă A și B sînt disjuncte.

În general, o cantitate declarată într-un bloc A este locală pentru acest bloc și globală pentru orice subbloc al blocului A, exceptînd pe cele în care este redeclarată.

În orice bloc se pot utiliza numai cantități globale sau locale.

**Exemplu**

Pentru structura de program :

```
B1 : begin real x, y ; ...
                L :
                ...
     B2 : begin real x, u ; ...
                    P :
                    ...
          B3 : begin array A [1 : 100] ;
                    ...
               end ;
               ...
          B4 : begin array B [1 : 100] ;
                    ...
               end ;
               ...
          end ;
     B5 : begin real x, w ; ...
                ...
             R :
             ...
          end ;
     end ;
```

dacă, prin L, G, N se notează respectiv, local, global, nedefinit, situația este următoarea :

| Blocul | cantitățile | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | x | y | L | u | P | a | b | w | r |
| B1 | L | L | L | N | N | N | N | N | N |
| B2 | L | G | G | L | L | N | N | N | N |
| B3 | G | G | G | G | G | L | N | N | N |
| B4 | G | G | G | G | G | L | L | N | N |
| B5 | L | G | G | N | N | N | N | L | L |



Scanned with OKEN Scanner

**Exemple de programe. 1.** Să se calculeze :

$$K=(n+1)\ C_n^3+pqC_{n+2}^p-C_{(n+2)^2}^{p^2}$$

folosind o procedură pentru calculul lui $C_x^y$.

Programul este următorul :

```
begin integer p, q, n, K ;
    integer procedure Comb (x, y, L) ;
                      value x, y : integer x, y ; label L ;
        begin integer c, i ;
              if x<y then go to L ; Comb : =1 ;
              if y>x−y then y : =x−y ;
              for i : =1 step 1 until y do
              Comb : = (Comb×(x−i+1))÷i
        end procedura Comb ;
p : = read ; q : = read ; n : = read ;
K : = (n+1)×Comb (n, 3, M1)×p×q×Comb (n+2, p, M2)−Comb ((n+2) ↑ 2,
p ↑ 2, M3) ;
print (K) ; go to STOP ;

M1 : print (N−A MERS PRIMUL APEL) go to STOP ;
M2 : print (N−A MERS AL DOILEA APEL) go to STOP ;
M3 : print (N−A MERS AL TREILEA APEL) go to STOP ;
STOP : end ;
```

**2.** Să se rezolve următoarele ecuații transcendente :

$x+\sin x-1=0$ cu aproximația $10^{-8}$ ;

$x+\sqrt{x}-1=0$ cu aproximația $10^{-9}\cdot 4$ ;

$\frac{1}{x}-\ln x-1=0$ cu aproximația $10^{-7}\cdot 2$,

folosind o procedură pentru rezolvarea ecuației $f(x)=0$ cu metoda înjumătățirii intervalului. [Se consideră că rădăcina este în intervalul $(a,\ b)$. ]

*Rezolvare*

```
begin real a, b, x ;
    real procedure f (x) ; value x ; real x ; f (x) : =x+sin (x)−1 ;
    real procedure g (x) ; value x ; real x ; g (x) : =x+sqrt (x)−1 ;
    rela procedure h (x) ; value x ; real x ; h (x) ; =1/x+ln (x)−1 ;
    real procedure calcradacina (f, a, b, epsilon) ;
        real procedure f ; value a, b ; real a, b, epsilon ;
        begin real A, B, c, C ; A : =f (a) ; B : =f (b) ;
        reluare : if abs (a−b)< epsilon go to E ;
                  c : =(a+b)/2 ; C : =f (c) ;
                  if A×C<0 then b : =c else a : =c ; go to reluare ;
              E : calcradacina : =a ;
        end procedura calcradacina ;
a : = read ; b : = read ;
x : = calcradacina (f, a, b, 10−8) ; print (x) ;
```



Scanned with OKEN Scanner

```
    a : =read ; b : = read ;
    x : = calcradacina (g, a, b, 4₁₀−9) ; print (x) ;
    a : = read ; b : = read ;
    x : = calcradacina (h, a, b, 2₁₀−7) ; print (x) ;
end ;
```

## 4.3. COMPILATORUL FLAG

Compilatorul FLAG (Fortran Load And Go) este destinat prelucrării programelor sursă FORTRAN, programe care trec prin fazele compilare şi apoi execuţie. Aşadar, nu se mai face apel la editorul de legături, încărcător, monitorul de înlănţuiri, compilarea făcîndu-se în memorie, iar editarea legăturilor făcînd-o compilatorul FLAG. Acest fapt a permis creşterea vitezei de compilare, faţă de compilatorul FORTRAN, de circa şase ori. Nu se apelează la serviciile SGF pentru memorarea şi prelucrarea tabelelor, toate informaţiile fiind prezente în memorie (compilatorul FLAG ocupă 40 K octeţi). Compatibilitatea totală cu compilatorul FORTRAN face să-i crească aria de aplicabilitate.

Cartelele de comandă specifice sînt următoarele :

```
.          COMPILE
% COMPILE
% ASSIGN
% LABEL
% RUN
* FETCH
* DEFINE FILE
* USED FILE
```

Cartela . COMPILE are structura :

. COMPILE FLAG [, $a_1$] [, $a_2$] ...

unde $a_1$, $a_2$, ... sînt argumente.

Sintaxa cartelei % COMPILE este :

% COMPILE FLAG [, $a_1$] [, $a_2$] ...

Continuările, dacă există, se fac după aceleaşi reguli ca la cartele de comandă numai că scrierea argumentelor se face începînd din coloana 10 a liniilor care constituie continuări.

După cum rezultă din definiţiile celor două cartele, argumentele sînt opţionale. În lipsa lor, compilatorul FLAG furnizează : programul sursă, lista eventualelor erori sintactice, un rezumat al compilării, programul generat, lista eventualelor erori apărute la editarea legăturilor FLAG.



Scanned with OKEN Scanner

Furnizarea unor informații suplimentare se poate dirija prin utilizarea argumentelor. Cele mai folosite argumente sînt :

BJM (Batch Job Mode), argument specific cartelei . COMPILE, prezența lui făcînd să nu mai fie luate în considerație alte argumente (dacă există, se iau drept comentarii), se utilizează atunci cînd o succesiune de programe sînt constituite ca o singură lucrare. Fiecare program este precedat de o cartelă % COMPILE.

NLG (No ListinG) anulează listarea programului sursă, la imprimantă apărînd numai * FETCH, * DEFINE FILE, * USED FILE, instrucțiunile greșite.

Opțiunile OBL, MAP, SEQ, DBL, ER, CDC, PCK, SRG, DBG au rol similar cu al celor din FORTRAN.

Parametru NXC (No eXeCute) are ca efect interzicerea execuției. Se utilizează pentru a face punerea la punct a sintaxei programelor.

MS : *n* (Memory Size) folosește la partajarea memoriei disponibile în două regiuni *P*, *V*. Regiunea *P* va cuprinde instrucțiunile generate, variabilele ce capătă valori la compilare, listele aferente instrucțiunilor FORMAT, tabelul rezumativ al fișierelor. *P* are cel mult 64 K octeți. Regiunea *V* este destinată variabilelor neinițializate (compilatorul le va inițializa cu zero), poate avea o lungime mai mare de 64 K octeți. În acest caz se împarte în zone de 64 K octeți, adresarea făcîndu-se în funcție de numărul zonei și de deplasarea în cadrul zonei. Pentru zona *V* se va rezerva $n \cdot 10\%$ din memoria disponibilă ($1 \leqslant n \leqslant 9$). Lipsa parametrului înseamnă $n=7$.

Cînd se lucrează în modul BJM este necesară prezența parametrului PN : < *nume* > (similar celui din cartela JOB) în cartela % COMPILE.

Celelalte cartele au structura similară cu cele utilizate în FORTRAN numai că în coloana 1 se scrie %, iar cuvîntul de desemnare începe din coloana 3.

În cele ce urmează sînt prezentate cîteva structuri de lucrări.

a) Compilarea și executarea unui singur program, fără fișiere.

```
.        JOB A,AN : 1977,PN : ALFA
.        COMPILE FLAG
    program sursă

% RUN
    date
.        EOJ
```

b) Compilarea și executarea unui singur program, cu fișiere

```
.        JOB B,AN : 1977,PN : BETA
.        INIT DV : RD1,VS : 111ABC
.        ALLOC DV : RD1, FN : 'DISC' ,AM : ANY,SZ : 14
.        COMPILE FLAG

DEFINE FILE X2 (DVT : RD RCF : D)=1
```



Scanned with OKEN Scanner

*program sursă*

```
% ASSIGN X,DV : RD1
% LABEL X, FN : 'DISC'
% RUN
        date
.       EOF
.       EOJ
```

c) Compilarea fără executare.

```
!           JOB C,AN : 1977,PN : GAMA
.           COMPILE FLAG,NXC
        program sursă
.           EOJ
```

d) Modul de lucru BJM

```
!           JOB D, AN : 1977, PN : DELTA
.           COMPILE FLAG, BJM
% COMPILE FLAG,PN : CIT1
        program sursă—1
% RUN
        date—1
% COMPILE FLAG,PN : CIT2
        program sursă—2
% RUN
        date—2
.....
% COMPILE FLAG,PN : CITN
        program sursă—N
% RUN
        date—N
.           EOJ
```

## 4.4. LIMBAJUL LISP

Este cunoscut faptul că pentru rezolvarea problemelor *nenumerice* la care scopul principal nu este calculul ci, de exemplu, demonstrarea unei teoreme, traducerea unui text, s-au elaborat o serie de limbaje. Din rîndul lor face parte limbajul LISP-FELIX, operațional pe calculatoare FELIX C-256.

4.4.1. **Elemente de bază.** Alfabetul limbajului cuprinde trei tipuri de caractere (simboluri) :

$< caracter > : : = < literă > | < cifră > | < caracter\ special >$



Scanned with OKEN Scanner

< *literă* > : : = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

< *cifră* > : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

< *caracter special* > : : = + | − | * | / | — | ( | . | )

Ca elemente de bază, limbajul FELIX-LISP cuprinde atomul (simbolul atomic) și *s-expresia*.

< *atom* > : : = < *atom nenumeric* > | < *atom numeric* >

< *atom nenumeric* > : : = <*literă*> | <*atom nenumeric*> | < *literă* > | < *atom nenumeric* > < *cifră* >

< *atom nenumeric* > : : = <*număr întreg*> | <*număr real*> | < *număr real* > E < *exponent* >

Observații. 1. Atomii nenumerici au maximum opt caractere.

2. Atomii numerici nu încep și nici nu se termină cu punct (marca zecimală).

3. Exponentul este număr întreg de cel mult două cifre, cu sau fără semn.

*S-expresiile* se definesc astfel :

< *s-expresie* > : : = < *atom* > | (< *s-expresie* >[, < *s-expresie* >])

O altă noțiune de bază este *lista* :

< *listă* > : : = (< *s-expresie* > ) | (< *s-expresie* > <*s-expresie* >) | (< *s-expresie* > · < *s-expresie* >)

**Exemple**

(X) echivalentă cu (X.NIL) NIL este atom special pentru a marca terminarea unei liste.

(X Y Z) echivalentă cu ((X. (Y.NIL)).(Z.NIL)).

() echivalentă cu NIL (lista vidă).

Observație. Mai multe spații consecutive echivalează cu unul singur.

În vederea efectuării prelucrării, LISP-FELIX dispune de trei tipuri de funcții :

– funcții sistem de tip subprogram ;
– funcții și predicate sistem definte ca *s-expresii* ;
– funcții utilizator definite ca *s-expresii*.

Numele funcțiilor de primele două tipuri constituie atomi rezervați, utilizarea lor în alt scop fiind semnalată ca eroare (tabelul 4.3 cuprinde lista atomilor rezervați).

*Constanta* este atom care se autodefinește. Orice atom numeric este o constantă. Unii dintre atomii nenumerici sînt și ei constante (de exemplu, NIL este o constantă).

O variabilă este un atom nenumeric care reprezintă un nume de dată.



Scanned with OKEN Scanner

*Tabelul 4.3*

Listă de atomi LISP-FELIX

| Atom | Tip | Atom | Tip | Atom | Tip |
|---|---|---|---|---|---|
| ADD1 | SUBR | FALSF | CONSTL | PLUS | SUBR |
| AND | SUBR | GREATERP | SUBR | PRINT | FSUBR |
| ATOM | SUBR | LENGTH | EXPR | QUOTE | FSUBR |
| BRANCH | FSUBR | LESSP | SUBR | QUOTIENT | SUBR |
| CAR | SUBR | LIST | SUBR | READ | FSUBR |
| CDR | SUBR | LOG | SUBR | REMAIND | SUBR |
| COND | FSUBR | MAX | SUBR | SEARCH | EXPR |
| CONS | SUBR | MIN | SUBR | SQRT | SUBR |
| COPY | EXPR | MINUS | SUBR | SUB1 | SUBR |
| CSETQ | FSUBR | MINUSP | SUBR | SUBLIS | EXPR |
| DEFINE | FSUBR | NIL | CONST | SUBST | EXPR |
| DELETE | EXPR | NOT | SUBR | SUBTRACT | SUBR |
| DIVITE | SUBR | NULL | SUBR | T | CONSTL |
| EQ | SUBR | NUMBERP | SUBR | TIMES | SUBR |
| EQUAL | EXPR | ONEP | SUBR | TRUE | CONSTL |
| EXPT | SUBR | OR | SUBR | VALUE | FSUBR |
| F | CONSTL | PAIR | EXPR | ZEROP | SUBR |

*Forma* este construită din constante și variabile fără a conta ordinea în lista de variabile. Calculul valorii unei forme se face pe baza unei liste de argumente. Există cîteva forme speciale cu acțiune subînțeleasă.

*Funcția* este o formă care depinde de mai multe variabile ordonate într-o listă (aceștia ar fi parametri formali). Cînd se calculează valoarea funcției, se dă o listă de argumente (parametri actuali). Aici, un rol important îl are notația LAMBDA, care se va prezenta ulterior. Orice funcție are un nume și argumente, fiind de forma

(< *atom nenumeric* > *a1 a2 ... am*)

Argumentele sînt *s-expresii* sau nume de alte funcții.

Faptul că o *s-expresie* este constantă se indică astfel :

(QUATE < *s-expresie* >).

4.4.2. *Structura unui program.* Orice program se compune din *secțiuni* (cel puțin una), o secțiune putînd avea trei componente : date, funcții utilizator, dubleți de interpretat (această componentă este obligatorie).

*Componenta de date* poate lipsi, aceasta fiind necesară numai atunci cînd trebuie ca unele variabile să capete valori (pentru a putea fi utilizate în dubleți).



Scanned with OKEN Scanner

Pentru atribuirea de valori nenumerice, respectiv numerice, se utilizează forma CSETQ care are respectiv formatele :

(CSETQ *x* (QUOTE < *s-expresie* >))
(CSETQ *y* *z*)

unde *x* este variabilă nenumerică, *y* este variabilă numerică, iar *z* este atom numeric (număr).

**Exemple**

```
(CSETQ MAR (QUOTE (X. Y. Z)))
(CSETQ ANA 0.0)
(CSETQ I 3.0E-1)
```

**Observație.** Valoarea unei variabile inițializată cu o formă CSETQ poate fi schimbată în componenta dubleți, cu formele speciale VALUE și READ.

*Funcțiile utilizator* se pot defini pe baza *funcțiilor sistem* (de tip subprogram sau definite). În acest scop se utilizează forma specială DEFINE cu structura :

```
DEFINE ((
(  (a1 (LAMBDA (lv1) d1))
   ..........................
   (an (LAMBDA (lvn) dn ))
))
```

unde *a1*, *a2*, ..., *an* sînt atomi (nume de funcții); *lv1*, *lv2*, ..., *lvn* sînt liste de variabile (parametri formali); *d1*, *d2*, ..., *dn* sînt definițiile funcțiilor (ca *s*-expresii).

**Exemple** Folosind funcțiile sistem PLUS și TIMES, funcțiile:

$$F(x)=3x$$
$$G(x)=x^3$$
$$H(x)=x^2+x+1$$

se definesc astfel :

```
DEFINE ((
    (F_(LAMBDA (X) (PLUS X X X)))
    (G_(LAMBDA (X) (TIMES X X)))
    (H_(LAMBDA (X) (PLUS (TIMES X X) X 1)))
))
```

*Componenta dubleți* este obligatorie. Forma generală a dubleților este

$$(f\_a1\_a2\_ \ldots \_an),$$

unde *f* este atom (nume de funcție), iar *a1*, *a2*, ..., *an* sînt argumentele funcției (parametri actuali). Ca argumente se pot lua constante, variabile,

funcţii. Dacă un argument este o variabilă, atunci aceasta se înlocuieşte cu ultima valoare temporară. În cazul că argumentul este o funcţie, aceasta este evaluată fiind considerată, la rîndul ei, ca un dublet. Funcţia se evaluează după ce toate argumentele ei sînt evaluate. Dacă funcţia este funcţie sistem de tip subprogram, se comunică valorile actuale ale argumentelor. În cazul că este vorba de o funcţie sistem de tip *s-expresie*, după ce se face corespondenţa dintre parametrii formali şi cei actuali, aceasta se tratează ca un dublet.

**Exemplu de dublet**
(PATRAT (PLUS A B))

*Formularul de programare.* Scrierea programelor se face în coloanele 1 – 72 ale formularului, în format liber. Nu este permisă scrierea mai multor *s-expresii* pe o linie. Este acceptată scrierea unei *s-expresii* pe mai multe rînduri, prezenţa caracterului * în coloana 72 marcînd liniile de continuare. În vederea scrierii corecte trebuie respectate regulile sintetizate în fig. 4.4, unde prin $x$ s-a notat un atom, iar ceea ce s-a haşurat sînt succesiuni nepermise (prin ␣ s-a marcat spaţiul).

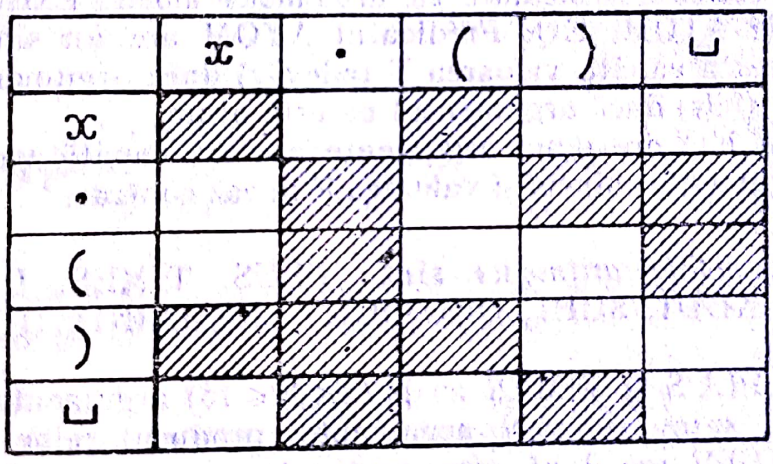



Fig. 4.4. Reguli de sitaxă pentru expresii LISP.

4.4.3. **Operaţii asupra structurilor** LISP-FELIX. Sistemul LISP-FELIX dispune de o serie de funcţii utilizabile ca subprograme predefinite, clasificate astfel : *funcţii elementare* cu argumente *s-expresii*, *predicate elementare* cu argumente atomi, *funcţii aritmetice*, *predicate numerice*, *funcţii logice*, alte funcţii şi forme.

4.4.3.1. *Funcţiile elementare* CAR, CDR, CONS. Funcţia CAR are un singur argument *s-expresie* care nu este atom şi are ca valoare prima componentă a *s-expresiei*.

De exemplu :

(CAR (QUOTE ((X.Y).Z))

conduce la valoarea X.Y în urma evaluării.

Funcția CDR este tot de un argument *s-expresie* nereductibilă la un atom și are ca valoare ceea ce rămîne din *s-expresie* prin înlăturarea primei componente.

De exemplu :

CDR (QUOTE (X.Y))

conduce la valoarea Y.

CONS este funcția care compune două *s-expresii*.

De exemplu :

(CONS (QUOTE X) (QUOTE Y))

conduce la *s-expresia* X.Y.

4.4.3.2. *Predicate elementare cu argumente atomi*. Există două astfel de predicate : ATOM, EQ. Predicatul ATOM are un singur argument *s-expresie*. Acesta capătă valoarea T (adevăr) dacă argumentul este atom și valoarea F (fals) dacă argumentul nu este atom.

Predicatul EQ are două argumente atomi ; capătă valoarea T dacă cele două argumente coincid și valoarea F în caz contrar.

4.4.3.3. *Funcțiile aritmetice* sînt : PLUS, TIMES, DIFF, MINUS, MAX, MIN, ADD1, SUB1, QUOTIENT, REMAIND, DIVIDE, SQRT, EXPT, LOG.

Funcțiile PLUS și TIMES au N ($2 \leqslant N \leqslant 15$) argumente, iar ca valori au respectiv, suma celor N argumente, produsul celor N argumente.

Funcția DIFF are două argumente. Are ca valoare diferența dintre primul și al doilea argument.

MINUS este funcție de un argument și are ca valoare argumentul cu semn schimbat.

Funcțiile MAX și MIN au N ($2 \leqslant N \leqslant 15$) argumente și sînt destinate calculului maximului, respectiv minimului.

Pentru a crește cu 1 valoarea unui argument se utilizează ADD1, iar pentru a micșora cu 1 valoarea unui argument se folosește SUB1.

Cîtul întreg a două numere se obține cu QUOTIENT, iar restul împărțirii întregi se obține cu funcția (de două argumente) REMAIND.

Funcția DIVIDE are două argumente, iar ca valoare are o listă de două componente : cîtul și restul împărțirii întregi dintre cele două argumente.



Scanned with OKEN Scanner

Pentru calculul radicalului de ordinul doi, pentru funcția exponențială și pentru funcția logaritmică se utilizează respectiv funcțiile SQRT, EXPT, LOG.

**Exemple**

(PLUS 5 −2)=3
(TIMES 5 3)=15
(DIFF 4 6)=−2
(MINUS −1)=1
(MAX 3 −2 4)=4
(MIN 3 −2 4)=−2
(ADD1 3)=4
(SUB1 3)=2
(QUOTIEND 10 3)=3
(REMAIND 10 3)=1
(DIVIDE 8 4)=(2 0)
(SQRT 9)=3
(EXPT 4 2)=$4^2$=16
(LOG 3 81)=$\text{Log}_3$ 81

4.4.3.4. *Predicatele numerice* sînt: ONEP, MINUSP, ZEROP, NUMBERP, LESSP, GREATERP. Primele patru sînt de o singură variabilă, iar ultimele două au două argumente. Valorile lor sînt definite astfel:

$$(\text{ONEP X})=\begin{cases}\text{T dacă X}=1\\ \text{F dacă X}\neq 1\end{cases}\qquad (\text{MINUSP X})=\begin{cases}\text{T dacă X}<0\\ \text{F dacă X}\geqslant 0\end{cases}$$

$$(\text{ZEROP X})=\begin{cases}\text{T dacă X}=0\\ \text{F dacă X}\neq 0\end{cases}\qquad (\text{NUMBER X})=\begin{cases}\text{T dacă X este atom numeric}\\ \text{F în caz contrar}\end{cases}$$

$$(\text{LESSP X Y})=\begin{cases}\text{T dacă X}<\text{Y}\\ \text{F dacă X}\geqslant\text{Y}\end{cases}\qquad (\text{GREATER X Y})=\begin{cases}\text{T dacă X}>\text{Y}\\ \text{F dacă X}\geqslant\text{Y}\end{cases}$$

4.4.3.5. *Funcții logice.* Acestea sînt AND, OR, EXOR și au N ($2\leqslant N\leqslant 15$) argumente. Funcția EXOR se definește astfel:

(EXOR F F)=F
(EXOR T T)=F
(EXOR T F)=T
(EXOR F T)=T

**Exemple**

(AND (ZEROP 4) (MINUSP −2))=F
(EXOR (OR (GREATERP 5 1) (ZEROP1)) T)=F

4.4.3.6. *Alte funcții și forme.* Funcția LIST are N ($2\leqslant N\leqslant 15$) argumente și are ca rol compunerea celor N argumente într-o listă:
(LIST (QUOTE X) (QUOTE (Y Z))=X (Y Z))
Predicatul NULL se definește astfel:

$$(\text{NULL X})=\begin{cases}\text{T dacă X este atomul NIL sau listă vidă}\\ \text{F în caz contrar}\end{cases}$$



Scanned with OKEN Scanner

Pentru facilităţi de programare există patru forme : VALUE, READ, PRINT, BRANCH.

*Forma* VALUE are una din structurile :

(VALUE *v s*) sau (VALUE *v d*) *v* fiind variabilă declarată în partea de date unde a avut o valoare iniţială, iar acum capătă o nouă valoare. Prin *s* s-a notat o *s-expresie*, iar prin *d* un dublet. Dacă, de exemplu, în partea de date a existat :

(CSETQ X (QUOTE (Y Z)))

atunci, în partea de dubleţi, se poate scrie, de exemplu :

(VALUE X (CDR Y Z T))

*Forma* READ :

(READ *v1 v2 ... vn*)

are rol de a da noi valori temporare unor variabile declarate în partea de date. Fiecare valoare rezultă de pe o cartelă de date (prin cartelă de date se înţelege o cartelă împreună cu continuările ei - dacă există). Pe cartele se dau *s-expresiile* din evaluarea cărora rezultă noile valori temporare.

*Forma* PRINT :

(PRINT *s-expresie*)

are ca efect listarea *s-expresiei* specificate ca argument.

O *formă* care nu există în alte variante LISP şi există la LISP-FELIX este BRANCH :

(BRANCH *a*) sau (BRANCH *d*)

unde *a* este atom numeric, iar *d* este dublet. Ea permite salt înainte sau înapoi peste atîţia dubleţi cît indică *a*, respectiv, *d*. Saltul este înainte dacă *a* (*d*) conduc la valori întregi pozitive şi înapoi dacă *a* (*d*) conduc la valori negative.

*Forma specială* COND :

(COND *a1*, *a2*, ... *an*),

unde argumentele sînt de structura (predicat *s-expresie*), are rol similar cu al instrucţiunilor de tip GO TO. Evaluarea se face astfel : valoarea lui COND este valoarea *s-expresiei* primului predicat cu valoarea T (parcurgerea argumentelor se face de la stînga la dreapta). Dacă nici un pedicat nu este T, programul se întrerupe.

**Exemplu.** Să se calculeze

$$S = \sum_{k=1}^{49} k \cdot (k+1)$$



Scanned with OKEN Scanner

Programul LISP-FELIX este :

```
(CSETQ S 0)
(CSETQ K 1)
(VALUE S (TIMES K (ADD1 K)))
(VALUE K (ADD1 K))
(BRANCH (COND ((ZEROP(DIFF 49 K) 1) (T −2))))
(PRINT S)
```

Pentru K=1, 2, . . ., 48 predicatul

```
ZEROP (DIFF 49 K)
```

este fals şi cum al doilea predicat al lui COND este T are efect întoarcerea (înapoi) cu doi dubleţi (valoarea −2 corespunzătoare acestui predicat T).
Cînd K=49 predicatul primului argument al lui COND este adevărat şi atunci se trece la dubletul următor (*s-expresia* primului argument al lui COND are valoarea 1), deci se tipăreşte rezultatul.

4.4.4. Structura unui job LISP. Un job are structura :

```
.       JOB X, AN : 1234, PN : IANA
.       COMPILE LISP
    program sursă LISP
.       EOJ
```

sau structura :

```
.       JOB X,AN : 4563,PN : MAR
.       FETCH LN : NAME, FN : LISP,DV : RD,VN : 1,GN :1
.       RUN
    date
.       EOJ
```

*Cartelele de comandă specifice* : TEST, RUN, FIN

*Cartela* TEST are structura

```
% TEST [NLG] [,NRP] [,DBL] [,MAP]
```

Parametrul NRP se utilizează cînd se doreşte numai aualiza sintaetică, fără generarea textului obiect.

*Cartela* RUN are următoarea sintaxă :

```
% RUN [TRACE]
```

Opţiunea TRACE provoacă scrierea rezultatelor intermediare.

Pentru marcarea sfîrşitului unei lucrări LISP şi trecerea controlului sistemului SIRIS 2/3 se utilizează cartela

```
% FIN
```



Scanned with OKEN Scanner

O lucrare se poate compune din faze. Structura unei faze este :

```
% TEST
   .
   .          secțiunea 1
   .
   END
% TEST
   .
   .          secțiunea 2
   .
   END
   .
  ..
   .
% TEST
   .
   .          secțiunea n
   .
   END
% RUN
```

Scanned with OKEN Scanner

# 5

# LIMBAJE PENTRU PROBLEME DE GESTIUNE

Trei limbaje destinate rezolvării problemelor de gestiune vor fi prezentate în cadrul acestui capitol : COBOL, LPG şi RPG.

## 5.1. LIMBAJUL COBOL

5.1.1. **Istoric.** Problema elaborării unui limbaj de programare pentru aplicaţiile de gestiune a fost discutată pentru prima dată la conferinţa organizată în mai 1959, la Washington. Comitetul conferinţei – CODASYL – propune crearea unui comitet pentru coordonarea lucrărilor de elaborare a limbajului. Acest comitet s-a întrunit prima oară în iunie 1959, stabilind un grup de lucru [73]. Sarcina grupului de lucru a constat în examinarea limbajelor comerciale existente (ca de exemplu, FLOWMATIC, AIMACO, COMTRAN etc.) şi stabilirea dacă unul dintre acestea ar putea fi extins şi adaptat scopului propus la conferinţa din mai 1959, sau dacă este necesară elaborarea unui nou limbaj [1]. Concluzia grupului de lucru a fost elaborarea unui nou limbaj, pentru care se propune numele COBOL (Common Business Oriented Language).

Pe baza specificaţiilor elaborate, în noiembrie 1959, este elaborată prima versiune a limbajului COBOL, care a fost aprobată de comitetul executiv al CODASYL în ianuarie 1960 şi publicată în aprilie 1960. Tot în 1960 a fost creat un comitet pentru dezvoltarea limbajului care, pe baza observaţiilor şi recomandărilor referitoare la prima versiune, elaborează a doua versiune, publicată în 1961 sub numele COBOL-61 (Revised Specifications for a Common Business Oriented Language). Versiunea următoare, publicată în 1962 sub numele COBOL-61 *Extended*, prezintă, în plus, facilităţile : editare de rapoarte şi sortare fişiere. Versiunile ulterioare – care au adus clarificări, modificări şi extensii versiunilor anterioare – au fost publicate în anii 1965, 1969, 1970 şi 1973, evoluţia limbajului fiind facilitată de publicaţia COBOL *Journal of Development*.



Scanned with OKEN Scanner

Activitatea de standardizare a limbajului a fost inițiată, în 1963, de către un comitet al ASA (American Standards Association), astăzi denumit ANSI (American National Standards Institute). Prima versiune standardizată – COBOL ANS (American National Standard) – a fost publicată în anul 1968, iar a doua, în 1973.

5.1.2. **Caracteristici.** Fiind foarte apropiat de limbajul natural (limba engleză), limbajul COBOL este accesibil unui cerc larg de utilizatori.

Prin caluzele de descriere și instrucțiunile de care dispune, limbajul COBOL permite definirea organizărilor clasice de fișiere (secvențiale, secvențiale-indexate, selective) implicate în rezolvarea problemelor de gestiune și a operațiilor prin care se realizează prelucrarea acestor fișiere (creare, consultare, actualizare, sortare). De asemenea, comparativ cu alte limbaje de programare, limbajul COBOL prezintă facilități suplimentare pentru definirea și realizarea de rapoarte. Totuși, prin setul de instrucțiuni aritmetice și de control, prin formatele diverse de reprezentare a datelor pe care le admite, prin posibilitatea definirii și prelucrării tablourilor, limbajul COBOL poate fi utilizat și în rezolvarea problemelor din alte domenii.

Termenul „common" sugerează faptul că programele sursă COBOL sînt, în general, compatibile, putînd fi compilate pe orice tip de calculator care are implementat un compilator COBOL.

În cele ce urmează este prezentată versiunea limbajului COBOL-ANS 1968, implementată pe calculatoarele de tip IRIS și FELIX ([117], [118]). În prezentare, un spațiu mai larg a fost atribuit facilităților limbajului (prelucrarea tablourilor, editarea rapoartelor) care nu au existat în versiunile implementate anterior pe aceste tipuri de calculatoare. Gramatica limbajului este definită prin intermediul metalimbajului CBL.

5.1.3. **Vocabularul limbajului COBOL** conține : caractere de bază, cuvinte cheie, identificatori, constante, numere de nivel și șabloane pentru descrierea datelor.

1) *Caracterele de bază*, în număr de 51, sînt împărțite în trei categorii :

– caracterele A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z și spațiu (sau blanc), numite *caractere alfabetice* ;

– caracterele 0, 1, 2, 3, 4, 5, 6, 7, 8 și 9, numite *caractere numerice* ;

– caracterele +, –, *, /, =, $, ,, ., ;, ' (sau "), (,), < și >, numite *caractere speciale.*

În limbajul COBOL se numește *caracter alfanumeric* orice caracter alfabetic, numeric sau special.

2) *Cuvintele cheie*, care au o semnificație bine precizată în cadrul limbajului, sînt cuvinte rezervate. Lista cuvintelor cheie este prezentată în tabelul 5.1.



Scanned with OKEN Scanner

*Tabelul 5.1*

## LISTA CUVINTELOR REZERVATE COBOL

ACCEPT
ACCESS
ACTUAL
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
BEFORE
BEGINNING
BLANK
BLOCK
BY
CALL
CF
CH
CHARACTERS
CHECKING
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMP
COMP-1
COMP-2
COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
CREATING
CSP
CURRENCY
C 01
C 02
C 03
C 04
C 05
C 06
C 07
C 08
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DECIMAL-POINT
DECLARATIVES
DELETE
DEPENDING
DESCENDING
DETAIL
DIRECT
DIRECT-ACCESS
DISPLAY
DIVIDE
DIVISION
DOWN
ELSE
END
ENDING
ENTER
ENTRY
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXCEEDS
EXIT
FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FORM-OVERFLOW
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HEADING
HIGH-VALUE
HIGH-VALUES
ID
IDENTIFICATION
IF
IN
INCLUDE
INDEX
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
I-O
I-O-CONTROL
IS
JUST
JUSTIFIED
KEY
LABEL
LABELS
LAST
LEADING
LEFT
LENGHT
LESS
LIMIT
LIMITS
LINE
LINE-COUNTER
LINES
LINKAGE
LOCATE
LOCK
LOW-VALUE



Scanned with OKEN Scanner

LOW-VALUES
MODE
MORE-LABELS
MOVE
MULTIPLE
MULTIPLY
NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OTHERWISE
OUTPUT
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POSITION
POSITIVE
PRINT
PRINTING
PRINT-SWITCH
PROCEDURE
PROCEED
PROCESSING
PROGRAM
PROGRAM-ID
QUOTE
QUOTES
RANDOM
RD
READ
RECORD
RECORDING
RECORDS
REDEFINES
REEL
RELEASE
REMAINDER
REMARKS
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RESTRICTED
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SELECT
SENTENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STOP
SUBTRACT
SUM
SUPPRES
SYMBOLIC
SYNC
SYNCHRONIZED
SYSIN
SYSIPT
SYSLST
SYSMAN
SYSOPT
SYSOUT
SYSPCH
SYSPUNCH
SYSRERUN
TALLY
TALLYING
TAPE
TERMINATE
TEXT
THAN
THEN
THROUGH
THRU
TIMES
TO
TRACK-AREA
TRACKS
TRANSFORM
TYPE
UNEQUAL
UNIT
UNIT-RECORD
UNITS
UNTIL
UP
UPON
USAGE
USE
USING
UTILITY
VALUE
VALUES
VARYING
WHEN
WITH
WORKING-STORAGE
WRITE
WRITE-ONLY
ZERO
ZEROES
ZEROS



Scanned with OKEN Scanner

3) *Identificatorii*, numiți în limbajul COBOL cuvinte utilizator, reprezintă succesiuni de caractere alfabetice (exceptînd spațiul), numerice și caracterul special – (liniuță) și sînt utilizați pentru referirea : programelor, datelor, structurilor de date, instrucțiunilor. În prezentarea construcțiilor limbajului, identificatorii vor fi desemnați prin notațiile *nume-program*, *nume-fișier*, *nume-dată* etc. Ca regulă generală, identificatorii au o lungime maximă de 30 de caractere, conțin cel puțin o literă, iar liniuța, dacă este utilizată, nu poate fi primul sau ultimul caracter. Excepții de la această regulă fac identificatorii asociați programelor, a căror lungime nu trebuie să depășească opt caractere, și cei asociați secvențelor de instrucțiuni, care pot fi formați și numai din cifre.

4) *Constantele* vocabularului COBOL sînt de două tipuri : *literale* și *constante figurative*.

*Literalele* pot reprezenta numere întregi sau reale, caz în care se numesc *numerice*, sau șiruri de caractere, caz în care se numesc *nenumerice*.

*Literalele numerice* pot fi exprimate :

– sub forma unei succesiuni de maximum 18 cifre (în care poate apărea și marca zecimală), precedată (opțional) de semnul + sau – ;

– sub forma : ± *mantisă* E ± *exponent* ; mantisa reprezintă o succesiune de maximum 16 cifre și trebuie să conțină marca zecimală, iar exponentul, un număr întreg format din una sau două cifre.

*Literalele nenumerice* reprezintă succesiuni de maximum 120 caractere EBCDIC delimitate prin apostrofuri (apostrofurile nu pot apărea în cadrul succesiunii).

*Constantele figurative* reprezintă un tip aparte de constante, specifice limbajului COBOL, identificate prin nume simbolice ce sînt cuvinte cheie. Valorile desemnate prin constantele figurative sînt date de semnificația cuvintelor cheie ce le-au fost asociate (vezi tabelul 5.2). Dacă o constantă figurativă este asociată unei date (de exemplu, prin instrucțiunea de transfer MOVE), lungimea constantei este considerată egală cu lungimea datei.

5) *Numerele de nivel*, utilizate în general pentru definirea ierarhiei datelor în cadrul unei structuri de date de tip articol, reprezintă numere formate de una sau două cifre, avînd valori cuprinse în intervalul [1, 49] sau egale cu 66, 77 sau 88.

Numărul de nivel 1 (sau 01) este asociat unei structuri de date de tip articol, iar numerele de nivel 2 pînă la 49 datelor ce compun o astfel de structură.

Celelalte numere de nivel au o utilizare particulară.

6) *Șabloanele* reprezintă succesiuni de maximum 30 de simboluri speciale (A, B, P, S, V, X, Z, 9, 0, , , . , +, –, CR, DB, * și simbolul monetar) utilizate pentru definirea categoriei și lungimii datelor.



Scanned with OKEN Scanner

*Tabelul 5.2*

**Constantele figurative**

| Constanta figurativă | Valoare |
|---|---|
| {ZERO / ZEROS / ZEROES} | unul sau mai multe caractere zero |
| {SPACE / SPACES} | unul sau mai multe caractere spațiu |
| {HIGH-VALUE / HIGH-VALUES} | unul sau mai multe caractere a căror reprezentare internă este $FF_{16}$ |
| {LOW-VALUE / LOW-VALUES} | unul sau mai multe caractere a căror reprezentare internă este $OO_{16}$ |
| {QUOTE / QUOTES} | unul sau mai multe caractere apostrof |
| ALL *'c'* | unul sau mai multe caractere desemnate prin *c* |

5.1.4. **Structura programelor.** Comparativ cu programele scrise în alte limbaje de programare, programele COBOL au o structură aparte, standardizată, în care apar cinci tipuri principale de unități structurale : *diviziuni*, *secțiuni*, *paragrafe*, *rubrici*, *fraze*. Caracteristica principală a programelor COBOL este structurarea lor în patru diviziuni : diviziunea de identificare (IDENTIFICATION DIVISION), diviziunea de echipament (ENVIRONMENT DIVISION), diviziunea de date (DATA DIVISION) și diviziunea de prelucrare (PROCEDURE DIVISION). Orice program COBOL trebuie să conțină aceste diviziuni ; ordinea în care acestea trebuie să apară în cadrul programului corespunde celei în care au fost enumerate.

1) *Diviziunea* IDENTIFICATION grupează informațiile necesare identificării programului (vezi tabelul 5.3) : numele programului, numele autorului, data scrierii programului, data compilării programului, precizări în legătură cu destinația și funcțiile programului. Cu excepția numelui programului care este obligatoriu, celelalte informații sînt opționale, fiind considerate de compilator comentarii.

2) *Diviziunea* ENVIRONMENT conține informații referitoare la :

– sistemul de calcul care realizează compilarea și execuția programului, caracterele care pun în evidență, la editarea datelor, marca zecimală și simbolul monetar – în secțiunea CONFIGURATION ;



Scanned with OKEN Scanner

*Tabelul 5.3*

**Diviziunea IDENTIFICATION**

| |
|---|
| {IDENTIFICATION / ID} DIVISION. |
| PROGRAM-ID. nume program. |
| [AUTHOR. paragraf comentariu.] |
| [INSTALLATION. paragraf comentariu.] |
| [DATE-WRITTEN. paragraf comentariu.] |
| [DATE-COMPILED. paragraf comentariu.] |
| [SECURITY. paragraf comentariu.] |
| [REMARKS. paragraf comentariu.] |

— caracteristicile fişierelor utilizate în program, tehnicile particulare utilizate pentru prelucrarea fişierelor, echipamentul necesar execuţiei programului — în secţiunea INPUT-OUTPUT (vezi tabelul 5.4).

*Tabelul 5.4*

**Diviziunea ENVIRONMENT**

| |
|---|
| ENVIRONMENT DIVISION. |
| CONFIGURATION SECTION. |
| [SOURCE-COMPUTER. paragraf-comentariu.] |
| [OBJECT-COMPUTER. paragraf-comentariu.] |
| [SPECIAL-NAMES. CURRENCY SIGN IS *literal* |
| DECIMAL-POINT IS COMMA.] |
| [INPUT-OUTPUT SECTION. |
| FILE-CONTROL. rubrici de descriere. |
| [I-O-CONTROL. rubrici de descriere.]] |

Aceste informaţii sînt specifice sistemului de calcul şi compilatorului utilizat.

3) *Diviziunea* DATA conţine descrierea datelor şi structurilor de date prelucrate prin execuţia programului. Diviziunea DATA poate fi structurată în patru secţiuni :

— secţiunea FILE, care apare în programele ce prelucrează fişiere şi conţine descrierea fişierelor şi articolelor ce le sînt asociate ;



Scanned with OKEN Scanner

– secţiunea WORKING-STORAGE, care apare în program numai dacă prelucrarea necesită definirea unor date care pot fi izolate (nestructurate) sau pot alcătui articole ;

– secţiunea LINKAGE, care apare numai în cadrul subprogramelor COBOL şi conţine descrierea parametrilor formali ;

– secţiunea REPORT, care apare în programele care utilizează editorul COBOL şi conţine descrierea structurii rapoartelor.

Descrierea fişierelor, rapoartelor, articolelor şi datelor se realizează prin intermediul *rubricilor de descriere*, care grupează una sau mai multe clauze.

4) *Diviziunea* PROCEDURE grupează instrucţiunile care descriu algoritmul prelucrării. Spre deosebire de secţiunile celorlalte diviziuni, care sînt identificate prin cuvinte cheie şi grupează informaţii prestabilite, în cazul diviziunii PROCEDURE identificarea şi conţinutul secţiunilor sînt stabilite de utilizator. Utilizarea secţiunilor în diviziunea PROCEDURE este, în general, opţională, fiind impusă numai de anumite situaţii : segmentarea programelor, sortarea fişierelor etc. Alte unităţi structurale întîlnite în cadrul diviziunii PROCEDURE sînt paragrafele şi frazele. Un *paragraf* conţine una sau mai multe fraze, iar o *frază* conţine una sau mai multe instrucţiuni COBOL, legate prin separatorii punct-virgulă, virgulă, spaţiul sau cuvîntul cheie THEN, şi se termină printr-un punct urmat de spaţiu.

O *instrucţiune* COBOL reprezintă un ansamblu de cuvinte (cuvinte cheie şi cuvinte utilizator) şi simboluri. Primul cuvînt este un cuvînt cheie (numit verb) care indică operaţia (operaţiile) descrisă prin instrucţiune.

5.1.5. **Descrierea datelor.** Atributele datelor sînt precizate prin intermediul unor construcţii standard ale limbajului COBOL, numite *clauze*. Numărul de nivel, numele şi clauza (clauzele) asociate unei date alcătuiesc o *rubrică de descriere*.

5.1.5.1. *Rubrici de descriere a datelor*. Există următoarele tipuri de rubrici de descriere a datelor :

– rubrici de descriere a datelor (elementare sau grupate) ce aparţin articolelor, identificate prin numere de nivel ce aparţin intervalului [01, 49] ;

– rubrici pentru gruparea, sub un nou nume, a datelor deja descrise, identificate prin numere de nivel 66 ;

– rubrici de descriere a datelor izolate, identificate prin numere de nivel 77 ;

– rubrici de descriere a numelor de condiţie, identificate prin numere de nivel 88.



Scanned with OKEN Scanner

Cu excepția rubricilor identificate prin numere de nivel 77, care pot apărea numai în secțiunile WORKING-STORAGE și LINKAGE, celelalte tipuri de rubrici pot apărea în oricare din secțiunile diviziunii DATA.

5.1.5.2. *Clauze pentru descrierea datelor.* În continuare sînt prezentate (cu excepția clauzei OCCURS, expusă în 5.1.10) clauzele ce pot fi utilizate în rubricile de descriere a datelor din secțiunile FILE, WORKING-STORAGE și LINKAGE. Clauzele specifice rubricilor ce definesc nume de condiție și cele specifice descrierii rapoartelor sînt prezentate în 5.1.7 și 5.1.12.

În tabelul 5.5 este prezentat formatul general al rubricii de descriere a datelor izolate și a celor ce aparțin articolelor (formatul 1), precum și formatul general al rubricii de grupare a datelor ce aparțin articolelor (formatul 2).

*Tabelul 5.5*

**Rubrica de descriere a datelor**

Format 1:

*număr-nivel* { *nume-dată-1* / FILLER }

[; REDEFINES *nume-dată-2*]

[; { PICTURE / PIC } IS *șablon*]

[; USAGE IS { DISPLAY / { COMPUTATIONAL / COMP } / { COMPUTATIONAL-1 / COMP-1 } / { COMPUTATIONAL-2 / COMP-2 } / { COMPUTATIONAL-3 / COMP-3 } / INDEX }]

[; { SINCHRONIZED / SYNC } { LEFT / RIGHT }]

[; { JUSTIFIED / JUST } RIGHT]

[; BLANK WHEN ZERO]

[; VALUE IS *literal*]

*Tabelul 5.5* (continuare)

```
[ ; OCCURS {intreg-1 TIMES                                          }]
           {intreg-2 TO intreg-3 TIMES DEPENDING ON nume-dată-3     }]
        [{ASCENDING }                                                ]
        [{DESCENDING}  KEY IS nume-dată-4 [, nume-dată-5] ...]       ]
        [INDEXED BY nume-index-1 [, nume-index-2] ... ]] .
```

Format 2 :

```
66 nume-dată-1 RENAMES nume-dată-2 [{THRU    } nume-dată-3 ].
                                    {THROUGH }
```

Pentru identificarea unei date în cadrul programului acesteia i se atribuie un nume (*nume-dată*-1) care, pentru a se evita calificarea, se stabileşte, în general, astfel încît să nu fie identic cu un alt identificator. În cazul în care este utilizată calificarea, numele atribuite datelor izolate şi articolelor definite în secţiunile WORKING-STORAGE şi LINKAGE trebuie să fie întotdeauna unice, deoarece acestea nu pot fi calificate. Cuvîntul cheie FILLER poate fi asociat unei date ca nume, atunci cînd în program nu se fac referiri la valorile acesteia.

Toate clauzele rubricii de descriere (format 1) sînt opţionale, iar ordinea în care sînt citate nu este impusă ; face excepţie clauza REDEFINES care trebuie să urmeze identificatorului *nume-data*-1/FILLER.

În cele ce urmează observaţiile făcute în legătură cu datele elementare se consideră valabile şi pentru datele izolate.

1) *Clauza* PICTURE defineşte categoria (tipul) şi lungimea unei date elementare. În limbajul COBOL există patru categorii de date : *alfabetice, numerice, numerice de editare* şi *alfanumerice* (tabelul 5.6). Semnificaţia simbolurilor ce pot figura în şabloanele de descriere a datelor este prezentată în tabelul 5.7.

— În legătură cu clauza PICTURE se pot face observaţiile prezentate în continuare.

1. Marca zecimală, indicată prin simbolul V, este virtuală, adică nu este reprezentată în memorie, dar este luată în consideraţie de compilator la generarea corectă a codului obiect.

2. Operaţia de editare are loc în momentul înregistrării valorilor datei în zona de memorie ce i-a fost rezervată şi constă în : inserarea semnului, a mărcii zecimale reale, înlocuirea zerourilor nesemnificative prin spaţii sau asteriscuri etc.

3. În mod obişnuit marca zecimală este indicată prin caracterul ., iar simbolul monetar prin caracterul $. Schimbarea acestor convenţii se poate realiza prin clauzele CURRENCY SIGN şi DECIMAL-POINT IS COMMA ale paragrafului SPECIAL-NAMES. Prima clauză stabileşte un alt simbol monetar (care nu poate fi însă unul dintre următoarele caractere : cifrele de la 0 la 9, literele A, B, C, D, E, P, R, S, V, X, Z sau caracterele speciale *, +, —, ., ,, (,), ' şi spaţiul, iar a doua clauză stabileşte drept marcă zecimală virgula. Alegerea virgulei ca marcă zecimală necesită ca în toate literalele numerice această convenţie să fie respectată.

*Tabelul 5.6*

**Categoriile de date în limbajul COBOL**

| Categoria datei | Valorile datei reprezintă : | Simboluri permise în şablonul de descriere | Lungimea datei (număr de caractere) | Operaţiunea clauzei USAGE | Formatul de reprezentare | Intervalul de valori | Lungimea zonei asociată datei (număr de locaţii) |
|---|---|---|---|---|---|---|---|
| alfabetică | succesiuni de caractere alfabetice | *A, B* | $1 \leqslant n < 65535$ | DISPLAY | Cod EBCIDIC | — | $n$ |
| alfanumerică | succesiuni de caractere EBCDIC | *A, X, 9, B, 0* | $1 \leqslant n < 65535$ | DISPLAY | Cod EBCDIC | — | $n$ |
| numerică de editare | numere pregătite pentru tipărire | *9, B, 0, ., , , +, —, CR, DB, *, Z*, simbolul monetar | $1 \leqslant n \leqslant 128$ | DISPLAY | Cod EBCDIC | — | $n$ |
| numerică | numere reale | *9, S, V* | $1 \leqslant n \leqslant 18$ | DISPLAY | zecimal-despachetat | $[-10^n+1, +10^n-1]$ | $n$ |
| | numere întregi | *9, S* | $1 \leqslant n \leqslant 4$ | COMP | cod complementar (semicuvînt) | $[-2^{15}, +2^{15}-1]$ | 2 |
| | | | $5 \leqslant n \leqslant 9$ | | cod complementar (cuvînt) | $[-2^{31}, +2^{31}-1]$ | 4 |
| | | | $10 \leqslant n \leqslant 18$ | COMP | cod complementar (dublu-cuvînt) | $[-2^{63}, +2^{63}-1]$ | 8 |
| | numere reale | — | $1 \leqslant n \leqslant 18$ | COMP-1 | virgulă mobilă simplă precizie | $[-0{,}72 \cdot 10^{76}, +0{,}72 \cdot 10^{76}]$ | 4 |
| | numere reale | — | $1 \leqslant n \leqslant 18$ | COMP-2 | virgulă mobilă dublă precizie | $[-0{,}72 \cdot 10^{76}, +0{,}72 \cdot 10^{76}]$ | 8 |
| | numere reale | *9, S, V* | $1 \leqslant n \leqslant 18$ | COMP-3 | zecimal împachetat | $[-10^n+1, +10^n-1]$ | $\left[\frac{n}{2}\right]+1$ |
| | adrese relative a elementelor tablourilor | — | $1 \leqslant n \leqslant 18$ | INDEX | cod complementar (cuvînt) | $[0, 2^{31}-1]$ | 4 |

Scanned with OKEN Scanner

*Tabelul 5.7*

**Simbolurile permise în şabloanele de descriere a datelor**

| Simbolul | Semnificaţie |
|---|---|
| A | un caracter alfabetic |
| B | poziţia în care va fi inserat caracterul spaţiu |
| 9 | o cifră |
| V | marca zecimală virtuală |
| S | valorile datei reprezintă numere cu semn |
| 0 | poziţia în care va fi inserat caracterul zero |
| Z | poziţia în care un zero nesemnificativ va fi înlocuit prin caracterul spaţiu |
| * | poziţia în care un zero nesemnificativ va fi înlocuit prin caracterul * |
| . | marca zecimală reală, dacă aceasta este punctul, sau poziţia în care va fi inserat caracterul . , dacă aceasta este virgula |
| , | marca zecimală reală, dacă aceasta este virgula, sau poziţia în care va fi inserat caracterul ,, dacă aceasta este punctul |
| + | poziţia în care va fi inserat caracterul +, dacă valoarea datei este un număr pozitiv, sau caracterul —, dacă valoarea datei este un număr negativ |
| — | poziţia în care va fi inserat caracterul spaţiu, dacă valoarea datei este un număr pozitiv, sau caracterul +, dacă valoarea datei este un număr negativ |
| CR/DB | poziţiile în care vor fi inserate două caractere spaţiu, dacă valoarea datei este un număr pozitiv, sau caracterele CR/DB, dacă valoarea datei este un număr negativ |
| $ (sau al simbol) | poziţia în care va fi inserat simbolul monetar |

2) *Clauza* USAGE defineşte formatul de reprezentare internă a valorilor unei date elementare sau grupate. Semnificaţia opţiunilor clauzei este prezentată în tabelul 5.6. Aşa cum rezultă din tabel, pentru datele descrise cu opţiunile COMP-1, COMP-2 şi INDEX nu este necesară clauza PICTURE.

O b s e r v a ţ i e. Clauzele PICTURE şi USAGE definesc tipul datelor. În funcţie de informaţiile furnizate prin aceste clauze, compilatorul COBOL alocă fiecărei date cantitatea de memorie necesară reprezentării valorilor sale. De menţionat că în cazul datelor descrise cu opţiunile COMP, COMP-1, COMP-2 şi INDEX, compilatorul rezervă zone de memorie de lungime de 2, 4 sau 8 locaţii, iar adresele acestora reprezintă multipli de 2, 4 sau 8 (tabelul 5.6). În cazul unei date descrise prin una din celelalte opţiuni, lungimea zonei este în funcţie de lungimea datei.



Scanned with OKEN Scanner

3) *Clauza* VALUE defineşte valoarea iniţială a unei date elementare sau grupate, descrisă în secţiunile WORKING-STORAGE sau REPORT.

Valoarea iniţială este indicată prin *literal*, care trebuie că coincidă ca tip cu tipul datei ; în cazul datelor elementare numerice, literarul trebuie să fie numeric sau constanta figurativă ZERO, iar în cazul datelor elementare nenumerice sau a datelor grupate, literalul trebuie să fie nenumeric sau una din constantele figurative : SPACES, HIGH-VALUE, QUOTE, ALL. De asemenea, lungimea literalului nu trebuie să depăşească lungimea datei.

4) *Clauza* JUSTIFIED indică un cadraj (aliniere) nestandard al valorilor datelor elementare alfabetice sau alfanumerice. În mod obişnuit, caracterele care alcătuiesc valoarea unei date elementare alfabetice sau alfanumerice sînt înregistrate în zona rezervată datei de la stînga la dreapta (vezi 5.1.8). Pentru datele descrise cu clauza JUSTIFIED, caracterele ce compun valorile acestor date vor fi înregistrate în zonă de la stînga la dreapta.

5) *Clauza* SYNCHRONIZED. Limbajul COBOL defineşte, în general, un model de organizare a memoriei pe caractere, în cazul datelor descrise cu opţiunile DISPIAY şi COMP-3, şi pe semicuvinte, cuvinte, duble-cuvinte în cazul datelor descrise cu opţiunile COMP, COMP-1 şi COMP-2. Prin intermediul clauzei SYNCHRONIZED, acest model de organizare poate fi modificat astfel încît şi datelor elementare descrise cu opţiunile DISPLAY şi COMP-3 să li se aloce, în funcţie de lungime, un anumit număr de cuvinte-memorie (lungimea zonelor rezervate reprezintă deci cel mai mic multiplu de 4, în care pot fi înregistrate caracterele ce compun valorile datei). În zonele rezervate, valorile datei sînt cadrate la dreapta (opţiunea RIGHT) sau la stînga (opţiunea LEFT), existînd posibilitatea rămînerii unor locaţii neutilizate, în cazul în care lungimea datei nu este multiplu de 4.

Clauza SYNCHRONIZED a fost introdusă în limbajul COBOL pentru compatibilitatea cu alte limbaje de programare (de exemplu, FORTRAN) care au un model de organizare a memoriei pe cuvinte.

6) *Clauza* BLANK WHEN ZERO poate fi citată numai în rubrica de descriere a unei date elementare numerice sau numerice de editare şi are ca efect înregistrarea, în fiecare locaţie a zonei rezervate acestei date, a caracterului spaţiu, dacă valoarea datei este egală cu zero ; dacă valoarea datei este diferită de zero, în zonă se înregistrează valoarea acesteia, inclusiv zerourile nesemnificative.

Datele descrise cu clauza BLANK sînt considerate numerice de editare, citarea lor ca operanzi aritmetici fiind interzisă.

7. *Clauza* REDEFINES permite ca aceeaşi zonă de memorie să fie alocată pentru două date elementare sau grupate (identificate prin

*nume-dată-1* și *nume-dată-2*) diferite. Structura și tipul acestor date pot diferi, însă lungimile lor trebuie să coincidă.

Facilitînd prelucrarea în aceeași zonă de memorie, în momente diferite, a unor date cu caracteristici diferite, clauza REDEFINES a fost introdusă, în general, pentru reutilizarea dinamică a zonelor de memorie. Datele desemnate prin *nume-dată-1* și *nume-dată-2* trebuie să aibă același număr de nivel (egal cu 77 sau cuprins în {1, 2, ..., 49}). De asemenea, rubrica în care este descrisă data *nume-dată-2* trebuie să urmeze celei în care este descrisă data *nume-dată-1*, iar între aceste două rubrici nu trebuie să apară decît, eventual, rubrici în care sînt descrise date subordonate datei *nume-dată-1*.

8) *Clauza* RENAMES permite atribuirea unui nume (*nume-dată-1*) unei date elementare sau grupate (*nume-dată-2*) sau unui grup de date elementare sau grupate (prima dată din grup fiind identificată prin *nume-dată-2*, iar ultima, prin *nume-dată-3*) care figurează consecutiv în cadrul descrierii unui articol. Datele *nume-dată-2* și *nume-dată-3* trebuie să aibă același număr de nivel (cu valori în {2,..., 49}) și nu pot fi elemente de tablouri. Rubricile de descriere în care apare clauza RENAMES trebuie să fie grupate după ultima rubrică de descriere a datelor articolului.

5.1.6. **Instrucțiuni aritmetice** În limbajul COBOL, descrierea operațiilor aritmetice se realizează prin cinci instrucțiuni: ADD, COMPUTE, DIVIDE, MULTIPLY și SUBTRACT (vezi tabelul 5.8). Operanzii citați în aceste instrucțiuni reprezintă date numerice elementare și literale

*Tabelul 5.8*

**Instrucțiuni aritmetice**

ADD

Format 1:

ADD { *nume-dată-1* / *literal-1* } [ , { *nume-dată-2* / *literal-2* } ] ... TO *nume-dată-m* [,*nume-dată-n*] ...
[ROUNDED] [; ON SIZE ERROR *instrucțiune* ... .]

Format 2:

ADD { *nume-dată-1* / *literal-1* } , { *nume-dată-2* / *literal-2* } [ , { *nume-dată-3* / *literal-3* } ] ... GIVING *nume-dată-n*
[ROUNDED] [ ; ON SIZE ERROR *instrucțiune* ... .]

Format 3:

ADD { CORRESPONDING / CORR } *nume-dată-1* TO *nume-dată-2*
[ROUNDED] [ ; ON SIZE ERROR *instrucțiune* ... .]



Scanned with OKEN Scanner

SUBTRACT

Format 1 :

SUBTRACT { *nume-dată-1* / *literal-1* } [ , { *nume-dată-2* / *literal-2* } ] ... FROM *nume-dată-n*
[ROUNDED] [ ; ON SIZE ERROR *instrucțiune* ... .]

Format 2 ;

SUBTRACT { *nume-dată-1* / *literal-1* } [ , { *nume-dată-2* / *literal-2* } ] ... FROM { *nume-dată-m* / *literal-m* } GIVING *nume-dată-n*
[ROUNDED] [ ; ON SIZE ERROR *instrucțiune* .... .]

Format 3 :

SUBTRACT { CORRESPONDING / CORR } *nume-dată-1* FROM *nume-dată-2*
[ROUNDED] [ ; ON SIZE ERROR *instrucțiune* ... .]

DIVIDE

Format 1 :

DIVIDE { *nume-dată-1* / *literal-1* } INTO *nume-dată-2*
[ROUNDED] [ ; ON SIZE ERROR *instrucțiune* ... .]

Format 2 :

DIVIDE { *nume-dată-1* / *literal-1* } INTO { *nume-dată-2* / *literal-2* } GIVING *nume-dată-3*
[ROUNDED] [REMAINDER *nume-dată-4*] [ ; ON SIZE ERROR *instrucțiune*... .]

Format 3 :

DIVIDE { *nume-dată-1* / *literal-1* } BY { *nume-dată-2* / *literal-2* } GIVING *nume-dată-3*
[ROUNDED] [REMAINDER *nume-dată-4*] [ ; ON SIZE ERROR *instrucțiune*... .]

MULTIPLY

Format 1 :

MULTIPLY { *nume-dată-1* / *literal-1* } BY *nume-dată-2*
[ROUNDED] [ ; ON SIZE ERROR *instrucțiune* ... .]



Scanned with OKEN Scanner

Format 2:

MULTIPLY { *nume-dată-1* / *literal-1* } BY { *nume-dată-2* / *literal-2* } GIVING *nume dată-3*
[ROUNDED] [; ON SIZE ERROR *instrucțiune* ... .]

COMPUTE

COMPUTE *nume-dată-1* [ROUNDED] = { *nume-dată-2* / *literal* / *expresie-aritmetică* }
[; ON SIZE ERROR *instrucțiune* ... .]

numerice. Reprezentarea internă a valorilor operanzilor poate fi oricare, întrucît compilatorul COBOL, prin apelul unor module din biblioteca sistemului, asigură conversiile necesare efectuării operațiilor. În cazul operanzilor cu zecimale, la generarea instrucțiunilor în limbaj-mașină corespunzătoare operațiilor descrise prin instrucțiunile aritmetice, compilatorul asigură alinierea acestora la marca zecimală.

Există două clauze opționale, specifice instrucțiunilor aritmetice: clauza ROUNDED, care indică rotunjirea valorii rezultatului obținut prin efectuarea unei operații aritmetice și clauza SIZE ERROR, care indică testarea depășirilor de capacitate.

1) *Instrucțiunile* ADD, SUBTRACT, DIVIDE, MULTIPLY. Principiul de execuție a acestor instrucțiuni este prezentat în tabelul 5.9, unde, prin $v_1$, $v_2$, ..., $v_i$, $v_m$, $v_n$ sînt notate valorile datelor *nume-dată-1/literal-1*, *nume-dată-2/literal-2* ..., iar prin $v_{1i}$ și $v_{2i}$, valorile celor $p$ date corespondente ($i = 1, 2, \ldots, p$) care aparțin datelor grupate *nume-dată-1* și *nume-dată-2* din formatul 3 al instrucțiunilor ADD și SUBTRACT. Se numesc *date corespondente* datele elementare ce aparțin unor date grupate, au același tip, același nume și același calificator.

În exemplul:

```
01 ART-1.                          01 ART-2.
   05 GRUP.                           05 GRUP.
      10 Y PIC 99.                       10 Y PIC 99.
      10 Z PIC x (35).                   10 W PIC 9 (4).
      10 W PIC 99.                      10 V PIC X (6).
      10 V PIC 9 (5).
   05 GRUP-1.                         05 GRUP-1.
      10 U PIC 99.                       10 U PIC 99.
      10 N PIC 9.                        10 N PIC A.
```

datele corespondente sînt Y, W și U.

*Tabelul 5.9*

**Operațiile descrise de instrucțiunile aritmetice**

| Instrucțiunea | Descrie operația de : | Format | Principiul de execuție |
|---|---|---|---|
| ADD | adunare | 1 | $v_m = v_1 + v_2 + \ldots + v_m$ și $v_n = v_1 + v_2 + \ldots + v_n$ |
| | | 2 | $v_n = v_1 + v_2 + \ldots + v_m$ |
| | | 3 | $v_{2i} = v_{1i} + v_{2i}$, $i = 1, 2, \ldots, p$ |
| SUBTRACT | scădere | 1 | $v_n = v_n - (v_1 + v_2 + \ldots + v_m)$ |
| | | 2 | $v_n = v_m - (v_1 + v_2 + \ldots + v_l)$ |
| | | 3 | $v_{2i} = v_{2i} - v_{1i}$, $i = 1, 2, \ldots, p$ |
| DIVIDE | împărțire | 1 | $v_2 = v_2 / v_1$ |
| | | 2 | $v_3 = v_2 / v_1$ și $v_4 = v_2 - (v_3 * v_1)$ |
| | | 3 | $v_3 = v_1 / v_2$ și $v_4 = v_1 - (v_3 * v_2)$ |
| MULTIPLY | înmulțire | 1 | $v_2 = v_1 * v_2$ |
| | | 2 | $v_3 = v_1 * v_2$ |

2) *Instrucțiunea* COMPUTE descrie operația prin care unei date (*nume-dată-1*) i se atribuie valoarea altei date (*nume-dată-2/literal*) sau valoarea unei expresii aritmetice.

În limbajul COBOL expresiile se formează din termeni, operatori și, eventual, paranteze. Termenii (operanzii) reprezintă date elementare numerice sau literale numerice. Operatorii sînt următorii : + (adunare), — (scădere), * (înmulțire), / (împărțire) și ** (ridicare la putere). Scrierea expresiilor aritmetice impune respectarea următoarelor reguli : operatorii trebuie să fie precedați și urmați de cel puțin un spațiu ; cel puțin un spațiu trebuie să preceadă o paranteză deschisă și să urmeze unei paranteze închise.

5.1.7. **Instrucțiuni de control.** Instrucțiunile pentru controlul secvenței execuției operațiilor în cadrul unui program COBOL sînt prezentate în tabelul 5.10.

*Tabelul 5.10*

**Instrucțiuni de control**

| |
|---|
| GO TO<br>Format 1 :<br>GO TO *nume-prelucrare*<br>Format 2 :<br>GO TO *nume-prelucrare-1* [,*nume-prelucrare-2*] ... [,*nume-prelucrare-n*]<br>DEPENDING ON *nume-dată* |

IF

IF *condiție* {*instrucțiune-1* / NEXT SENTENCE} ; ELSE {*instrucțiune-2* / NEXT SENTENCE}

EXIT

EXIT [PROGRAM].

STOP

STOP {*literal* / RUN}

CALL

CALL *nume-program* [USING *nume-dată-1* [,*nume-dată-2*] ...]

PERFORM

Format 1:
PERFORM *nume-prelucrare-1* [THRU *nume-prelucrare-2*]

Format 2:
PERFORM *nume-prelucrare-1* [THRU *nume-prelucrare-2*] {*nume-dată-1* / *intreg-1*} TIMES

Format 3:
PERFORM *nume-prelucrare-1* [THRU *nume-prelucrare-2*] UNTIL *condiție-1*

Format 4:
PERFORM *nume-prelucrare-1* [THRU *nume-prelucrare-2*]

VARYNG {*nume-dată-1* / *nume-index-1*} FROM {*nume-dată-2* / *nume-index-2* / *literal-2*} BY {*nume-dată-3* / *literal-3*} UNTIL *condiție-1*

[AFTER {*nume-dată-4* / *nume-index-4*} FROM {*nume-dată-5* / *nume-index-5* / *literal-5*} BY {*nume-dată-6* / *literal-6*} UNTIL *condiție-2*]

[AFTER {*nume-dată-7* / *nume-index-7*} FROM {*nume-dată-8* / *nume-index-8* / *literal-8*} BY {*nume-dată-9* / *literal-9*} UNTIL *condiție-3*]

1) *Instrucțiunea* GO TO descrie o operație de salt, adică specifică instrucțiunea cu care va continua execuția programului. Această instrucțiune, care trebuie să fie prima instrucțiune a unui paragraf sau secțiuni din diviziunea PROCEDURE, este identificată printr-o etichetă, numită de limbajul COBOL *nume de prelucrare.*

F o r m a t u l 1 descrie un salt necondiționat la instrucțiunea identificată prin *nume-prelucrare.*



Scanned with OKEN Scanner

Formatul 2 descrie un salt condiţionat, instrucţiunea cu care va continua execuţia programului fiind stabilită prin testarea valorii datei *nume-dată* (această valoare trebuie să reprezinte un număr întreg pozitiv cuprins între 1 şi *n*) : saltul se va produce la instrucţiunea identificată prin numele de prelucrare al cărui număr de ordine, în cadrul instrucţiunii, coincide cu valoarea datei *nume-data.*

În cazul în care valoarea datei *nume-data* nu este cuprinsă între 1 şi *n*, execuţia programului continuă cu instrucţiunea ce urmează instrucţiunii GO TO.

2) *Instrucţiunea* IF descrie următoarele operaţii :

– testarea unei condiţii şi stabilirea valorii logice (adevărat sau fals) a acesteia ;

– execuţia instrucţiunii (instrucţiunilor) desemnată prin *instrucţiune-1*, dacă valoarea logică a condiţiei este adevărat ;

– execuţia instrucţiunii (instrucţiunilor) desemnată prin *instrucţiune-2*, dacă valoarea logică a condiţiei este fals ;

– trecerea la execuţia primei instrucţiuni a frazei următoare, dacă este citată opţiunea NEXT SENTENCE sau dacă ia sfîrşit execuţia instrucţiunilor desemnate prin *instrucţiune-1* sau *instrucţiune-2* (opţiunea ELSE NEXT SENTENCE poate fi omisă dacă precede punctul ce marchează sfîrşitul frazei).

În cazul în care *instrucţiune-1* şi/sau *instrucţiune*-2 reprezintă instrucţiuni IF, perechile IF, ELSE se asociază ca perechile de paranteze (,) într-o expresie aritmetică.

Prin testarea unei condiţii, numită în limbajul COBOL şi *test*, se poate determina : relaţia dintre doi termeni (test de relaţie) ; clasa unei date (test de clasă) ; valoarea algebrică a unei date (test de semn) ; dacă valoarea unei date este egală cu o valoare dată sau aparţine unei mulţimi de valori dată (test de nume de condiţie).

*Testul de relaţie* stabileşte dacă valoarea unui termen este mai mică, egală sau mai mare decît valoarea celuilalt termen. Formatul condiţiei este :

$$\left\{\begin{array}{l}\textit{nume-dată-1}\\ \textit{nume-dată-index-1}\\ \textit{nume-index-1}\\ \textit{literal-1}\\ \textit{expresie-aritmetică-1}\end{array}\right\} \text{IS } [\underline{\text{NOT}}] \left\{\begin{array}{l}\underline{\text{GREATER}}\ \underline{\text{THAN}} \quad (>)\\ \underline{\text{LESS}}\ \underline{\text{THAN}} \quad (<)\\ \underline{\text{EQUAL}}\ \underline{\text{TO}} \quad (=)\end{array}\right\} \left\{\begin{array}{l}\textit{nume-dată-2}\\ \textit{nume-dată-index-2}\\ \textit{nume-index-2}\\ \textit{literal-2}\\ \textit{expresie-aritmetică-2}\end{array}\right\}$$

Valoarea logică a unui test de relaţie este adevărat, dacă termenii comparaţi se află în relaţia indicată de operatorii de relaţie, şi fals, în caz contrar. Dacă operatorul de relaţie este precedat de operatorul logic



Scanned with OKEN Scanner

NOT, valoarea logică a testului este adevărat, dacă cei doi termeni nu sînt în relaţia indicată de operatorul de relaţie, şi fals, în caz contrar; utilizarea operatorului logic NOT suplineşte astfel lipsa operatorilor de relaţie $\leqslant$, $\geqslant$ şi $\neq$.

Compararea termenilor se realizează, în principiu, astfel:

– în cazul în care ambii termeni sînt numerici, se compară valorile algebrice ale acestora (valorile termenilor pot avea reprezentări interne diferite);

– în cazul în care cel puţin un termen este nenumeric, se compară succesiv, de la stînga la dreapta, caracterele ce compun valorile termenilor (dacă numărul caracterelor termenilor nu este egal, compararea este precedată de completarea, la dreapta, cu caractere spaţiu, a valorii termenului cu lungime mai mică); dacă unul din termeni este numeric, iar valoarea sa este reprezentată zecimal-împachetat, în cod complementar sau în virgulă mobilă, compararea nu este precedată de conversia acestei valori în format zecimal-despachetat.

*Testul de clasă* stabileşte dacă valoarea unei date este numerică (formată din caractere numerice) sau alfabetică (formată din caractere alfabetice). Formatul condiţiei este

*nume-dată* IS [NOT] $\left\{\begin{matrix}\text{NUMERIC}\\ \text{ALPHABETIC}\end{matrix}\right\}$

Valoarea logică a testului de clasă este adevărat, dacă:

– valoarea unei date elementare numerice (reprezentată zecimal-împachetat sau zecimal-despachetat), alfanumerice sau a unei date grupate este formată numai din cifre;

– valoarea unei date elementare nenumerice sau a unei date grupate este formată numai din litere şi caracterul spaţiu.

*Testul de semn* stabileşte dacă valoarea algebrică a unei date elementare numerice de calcul sau a unei expresii aritmetice este mai mică, egală sau mai mare decît zero. Formatul condiţiei este

$\left\{\begin{matrix}\textit{nume-dată}\\ \textit{expresie-aritmetică}\end{matrix}\right\}$ IS [NOT] $\left\{\begin{matrix}\text{NEGATIVE}\\ \text{POSITIVE}\\ \text{ZERO}\end{matrix}\right\}$

*Testul numelui de condiţie* reprezintă o facilitate a limbajului COBOL care simplifică exprimarea unei condiţii. Un nume de condiţie este un identificator asociat unei valori sau unei mulţimi de valori ale unei date şi este definit printr-o rubrică de forma

88 *nume-condiţie* $\left\{\begin{matrix}\text{VALUE IS}\\ \text{VALUE ARE}\end{matrix}\right\}$ $\left\{\textit{literal-1}\left[\left\{\begin{matrix}\text{THROUGH}\\ \text{THRU}\end{matrix}\right\}\textit{literal-2}\right]\right\}\ldots$

Această rubrică trebuie să urmeze celei în care este descrisă data a cărei valoare este testată.

În instrucțiunea IF condiția este indicată prin numele de condiție. Valoarea logică a unui test de nume de condiție este adevărat dacă valoarea datei aparține valorii sau mulțimii de valori asociate numelui de condiție prin *literal-2*.

Condițiile prezentate reprezintă *condiții simple*. Unite prin operatorii logici AND și/sau OR și, eventual, grupate între paranteze, condițiile simple pot forma *condiții compuse*. În absența parantezelor, ierarhia operatorilor care apar în cadrul unei condiții compuse este: operatorii aritmetici, operatorii de relație, operatorul NOT, operatorii logici AND și OR. Valoarea logică a unei condiții compuse poate fi, ca și în cazul condițiilor simple, adevărat sau fals.

3) *Instrucțiunea* PERFORM descrie, în ordine, următoarele operații:

— saltul la prima instrucțiune a unei secvențe de prelucrare care grupează unul sau mai multe paragrafe (sau secțiuni) succesive;

— execuția necondiționată sau condiționată, de un anumit număr de ori, a acestei secvențe;

— saltul (revenirea) la instrucțiunea care urmează instrucțiunii PERFORM.

Primul paragraf (sau secțiune) al secvenței de prelucrare este identificat prin *nume-prelucrare-1*, iar ultimul, prin *nume-prelucrare-2*.

F o r m a t 1. Secvența de prelucrare este executată, necondiționat, o singură dată.

F o r m a t 2. Secvența de prelucrare este executată de un număr de ori, egal cu valoarea datei *nume-dată-1* sau *literal-1*, care trebuie să reprezinte un număr întreg pozitiv, diferit de zero. Dacă această valoare este negativă, sau egală cu zero, execuția secvenței de prelucrare este abandonată (execuția programului continuînd cu instrucțiunea ce urmează instrucțiunii PERFORM). Modificarea valorii datei *nume-dată-1* prin instrucțiuni ale secvenței de prelucrare nu afectează numărul de execuții ale acesteia.

F o r m a t 3. Secvența de prelucrare este executată pînă în momentul în care valoarea logică a condiției este adevărat. Execuția instrucțiunilor secvenței de prelucrare este precedată de testarea condiției; dacă valoarea logică a acesteia este fals, acestea nu mai sînt executate, controlul fiind transferat instrucțiunii ce urmează instrucțiunii PERFORM.

F o r m a t u l 4 al instrucțiunii este prezentat în 5.1.10.

4) *Instrucțiunea* EXIT definește un punct de ieșire (revenire) dintr-un subprogram COBOL sau dintr-o secvență de prelucrare apelată prin instrucțiunile PERFORM sau SORT. Instrucțiunea EXIT trebuie să

Această rubrică trebuie să urmeze celei în care este descrisă data a cărei valoare este testată.

În instrucțiunea IF condiția este indicată prin numele de condiție. Valoarea logică a unui test de nume de condiție este adevărat dacă valoarea datei aparține valorii sau mulțimii de valori asociate numelui de condiție prin *literal-2*.

Condițiile prezentate reprezintă *condiții simple*. Unite prin operatorii logici AND și/sau OR și, eventual, grupate între paranteze, condițiile simple pot forma *condiții compuse*. În absența parantezelor, ierarhia operatorilor care apar în cadrul unei condiții compuse este: operatorii aritmetici, operatorii de relație, operatorul NOT, operatorii logici AND și OR. Valoarea logică a unei condiții compuse poate fi, ca și în cazul condițiilor simple, adevărat sau fals.

3) *Instrucțiunea* PERFORM descrie, în ordine, următoarele operații:

— saltul la prima instrucțiune a unei secvențe de prelucrare care grupează unul sau mai multe paragrafe (sau secțiuni) succesive;

— execuția necondiționată sau condiționată, de un anumit număr de ori, a acestei secvențe;

— saltul (revenirea) la instrucțiunea care urmează instrucțiunii PERFORM.

Primul paragraf (sau secțiune) al secvenței de prelucrare este identificat prin *nume-prelucrare-1*, iar ultimul, prin *nume-prelucrare-2*.

Format 1. Secvența de prelucrare este executată, necondiționat, o singură dată.

Format 2. Secvența de prelucrare este executată de un număr de ori, egal cu valoarea datei *nume-dată-1* sau *literal-1*, care trebuie să reprezinte un număr întreg pozitiv, diferit de zero. Dacă această valoare este negativă, sau egală cu zero, execuția secvenței de prelucrare este abandonată (execuția programului continuînd cu instrucțiunea ce urmează instrucțiunii PERFORM). Modificarea valorii datei *nume-dată-1* prin instrucțiuni ale secvenței de prelucrare nu afectează numărul de execuții ale acesteia.

Format 3. Secvența de prelucrare este executată pînă în momentul în care valoarea logică a condiției este adevărat. Execuția instrucțiunilor secvenței de prelucrare este precedată de testarea condiției; dacă valoarea logică a acesteia este fals, acestea nu mai sînt executate, controlul fiind transferat instrucțiunii ce urmează instrucțiunii PERFORM.

Formatul 4 al instrucțiunii este prezentat în 5.1.10.

4) *Instrucțiunea* EXIT definește un punct de ieșire (revenire) dintr-un subprogram COBOL sau dintr-o secvență de prelucrare apelată prin instrucțiunile PERFORM sau SORT. Instrucțiunea EXIT trebuie să



Scanned with OKEN Scanner

formeze singură un paragraf ; opțiunea PROGRAM poate fi citată numai dacă instrucțiunea apare în subprograme COBOL.

5) *Instrucțiunea* STOP descrie operația de oprire temporară (prin opțiunea *literal*) sau definitivă (prin opțiunea RUN) a execuției programului. În primul caz, literalul — a cărui lungime nu poate depăși 72 de caractere — este tipărit la mașina de scris, iar execuția programului poate fi reluată, de la pupitrul de comandă, prin apăsarea tastei NEW-LINE. În cazul al doilea, execuția programului este încheiată, iar controlul este redat sistemului de operare.

6) *Subprograme* COBOL — *instrucțiunea* CALL. Subprogramele COBOL au, în principiu, aceeași structură ca și programele COBOL obișnuite. Deosebirile constau în prezența, în cadrul subprogramelor, a secțiunii LINKAGE, a unor clauze și instrucțiuni ce definesc parametrii transmiși acestora, punctele de intrare/ieșire în/din subprogram.

Comparativ cu subprogramele scrise în alte limbaje de programare, subprogramele COBOL au un singur punct de intrare (care este prima instrucțiune a subprogramului), iar transmiterea parametrilor se realizează prin adrese.

Execuția unui subprogram COBOL este cerută din cadrul unui program sau subprogram COBOL, numit *program apelant*, prin intermediul instrucțiunii CALL (tabelul 5.9). În formatul instrucțiunii, *nume-program* indică numele subprogramului, iar clauza USING indică, prin *nume-dată-1*, *nume-dată-2*, ... parametrii actuali transmiși subprogramului. Parametrii reprezintă date elementare sau articole definite în diviziunea DATA a programului apelant, în rubrici de descriere cu numere de nivel 77 sau 01.

Comparativ cu un program principal, un subprogram COBOL prezintă următoarele particularități :

— parametrii formali sînt specificați în clauza USING din titlul diviziunii PROCEDURE, în aceeași ordine în care au fost specificați parametrii actuali în clauza USING a instrucțiunii CALL, prin aceleași nume sau prin nume diferite ;

— descrierea parametrilor formali se realizează în secțiunea LINKAGE și trebuie să fie echivalentă cu descrierea făcută în programul apelant parametrilor actuali corespondenți ;

— revenirea din subprogram în programul apelant este indicată prin instrucțiunea EXIT PROGRAM.

5.1.8. **Instrucțiuni de transfer și transformare.** Formatele acestor instrucțiuni sînt prezentate în tabelul 5.11.

1) *Instrucțiunea* MOVE descrie operația prin care uneia sau mai multor date li se atribuie valoarea altei date. Zona corespunzătoare datei a cărei valoare este atribuită se numește *zonă emițătoare*, iar zonele



Scanned with OKEN Scanner

*Tabelul 5.11*

**Instrucțiuni de transfer și transformare**

MOVE

Format 1 :

MOVE { *nume-dată-1* / *literal* } TO *nume-dată-2* [, *nume-dată-3*] ...

Format 2 :

MOVE { CORRESPONDING / CORR } *nume-dată-1* TO *nume-dată-2*

EXAMINE

Format 1 :

EXAMINE *nume-dată-1* TALLYING { ALL / LEADING / [UNTIL] FIRST } { *nume-dată-2* / *literal-1* }

[ REPLACING BY { *nume-dată-3* / *literal-2* } ]

Format 2 :

EXAMINE *nume-dată-1* REPLACING { ALL / LEADING / [UNTIL] FIRST } { *nume-dată-2* / *literal-1* } BY { *nume-dată-3* / *literal-2* }

TRANSFORM

TRANSFORM *nume-dată* CHARACTERS FROM { *nume-dată-1* / *literal-1* / *constantă-figurativă-1* } TO { *nume-dată-2* / *literal-2* / *constantă-figurativă-2* }

ACCEPT

Format 1 :

ACCEPT *nume-dată*

Format 2 :

ACCEPT *nume-dată* FROM CONSOLE

Format 3 :

ACCEPT *nume-dată* FROM { DATE / DAY / TIME }



Scanned with OKEN Scanner

DISPLAY

Format 1:

DISPLAY { *nume-dată-1* / *literal-1* } [ , { *nume-dată-2* / *literal-2* } ] ...

Format 2:

DISPLAY { *nume-dată-1* / *literal-1* } [ , { *nume-dată-2* / *literal-2* } ] ... UPON CONSOLE

Format 3:

DISPLAY { *nume-dată-1* / *literal-1* } [ , { *nume-dată-2* / *literal-2* } ] ... UPON SYSPUNCH

corespunzătoare datelor cărora li se atribuie valoarea se numesc *zone receptoare.*

În limbajul COBOL operația de atribuire este denumită, în mod obișnuit, *transfer.*

Formatul 1 definește transferul valorii datei *nume-dată-1/literal* (care poate fi și o constantă figurativă) în zonele asociate datelor *nume-dată-2*, *nume-dată-3*, ...

Formatul 2 definește transferul valorilor datelor corespondente (vezi 5.1.6) ce aparțin datei grupate *nume-dată-1* în zonele asociate datelor corespondente ce aparțiu datei grupate *nume-dată-2.*

În funcție de tipul datelor citate în instrucțiunea MOVE, transferul poate fi:

– *numeric,* dacă data corespunzătoare zonei receptoare este o dată elementară numerică sau numerică de editare;

– *alfanumeric,* dacă data corespunzătoare zonei receptoare este o dată elementară alfabetică, alfanumerică sau dacă data corespunzătoare zonei emițătoare este o dată grupată.

*Transferul numeric* se efectuează după următoarele reguli:

– caracterele ce corespund valorii de transferat sînt înregistrate în zona receptoare de la marca zecimală spre stînga și, eventual, spre dreapta (dacă valoarea transferată reprezintă un număr cu zecimale);

– dacă numărul caracterelor de transferat este mai mic decît numărul locațiilor zonei receptoare, în fiecare locație neocupată prin transfer se înregistrează caracterul zero; în caz contrar, caracterele ce nu pot fi înregistrate sînt neglijate;

– dacă formatul de reprezentare al valorilor în zona receptoare este diferit de cel al valorii din zona emițătoare, transferul este precedat de

conversia valorii de transferat la formatul de reprezentare corespunzător zonei receptoare ;

— dacă data corespunzătoare zonei receptoare este numerică de editare, prin transfer se realizează și editarea, conform clauzei PICTURE.

În principiu, transferul numeric impune ca data corespunzătoare zonei emițătoare să fie o dată elementară numerică.

*Transferul alfanumeric* se efectuează (în absența clauzei JUSTIFIED în rubrica de descriere a datei ce corespunde zonei receptoare) după următoarele reguli :

— caracterele ce corespund valorii de transferat sînt înregistrate în zona receptoare de la stînga spre dreapta ;

— dacă numărul caracterelor de transferat este mai mic decît numărul locațiilor zonei receptoare, în fiecare locație neocupată prin transfer se înregistrează caracterul spațiu ; în caz contrar, caracterele ce nu pot fi înregistrate sînt neglijate ;

— dacă valoarea din zona emițătoare este reprezentată în virgulă mobilă, în cod complementar sau în zecimal-împachetat, transferul este precedat de conversia valorii transferate în virgulă mobilă format extern (± *mantisă* E ± *exponent* ; mantisa este formată din 7 sau 17 cifre, după cum reprezentarea internă este în virgulă mobilă simplă sau dublă precizie, iar exponentul este format din două cifre), în primul caz, și în zecimal-despachetat, în ultimele două cazuri.

Observație. Dacă cel puțin una din datele citate în instrucțiunea MOVE nu este o dată elementară, transferul este considerat alfanumeric. Toate constantele figurative sînt considerate literale nenumerice ; constanta figurativă ZERO este asimilată și literalelor numerice. Citarea unei constante figurative în instrucțiunea MOVE implică înregistrarea, în fiecare locație a zonei receptoare, a caracterului desemnat prin numele constantei.

2) *Instrucțiunea* ACCEPT descrie operația de transfer în zona corespunzătoare datei *nume-dată* a :

— valorii unei date înregistrată pe o cartelă, citită prin intermediul unității standard de intrare a sistemului, în cazul formatului 1 ;

— caracterelor furnizate de operator la mașina de scris, în cazul formatului 2 ;

— datei curente (sub forma *zzllaa* — dacă este utilizată opțiunea DATE — sau sub forma *zzzaa* — dacă este utilizată opțiunea DAY ; *aa* reprezintă ultimele două cifre ale anului, *ll*, luna, iar *zzz* numărul zilei în cadrul anului) sau a orei curente (sub forma *hh mm ss cc*, dacă este utilizată opțiunea TIME ; *hh* reprezintă ora, *mm*, minutul, *ss*, secunda, iar *cc* sutimea de secundă a orei curente), în cazul formatului 3.

Numărul caracterelor transferate nu trebuie să depășească 80 și, respectiv, 72 în cazul utilizării primelor două formate ; în cazul utilizării formatului 3, pentru înregistrarea datei sau a orei curente, trebuie



Scanned with OKEN Scanner

să se rezerve, în funcție de opțiunea utilizată, o zonă de memorie de 6, 5 și, respectiv, 8 locații.

3) *Instrucțiunea* DISPLAY descrie operația de transfer a valorii datelor *nume-dată-1/literal-1* (care poate fi și o constantă figurativă, cu excepția lui ALL) ... și înregistrarea lor consecutivă;

– pe suportul unității standard de ieșire a sistemului, în cazul formatului 1;

– la mașina de scris, în cazul formatului 2;

– pe suportul unității standard de perforare a sistemului, în cazul formatului 3.

Execuția instrucțiunii determină, în funcție de formatul utilizat, tipărirea unui rînd la imprimantă, la mașina de scris sau perforarea unei cartele; numărul caracterelor înregistrate pe un rînd sau pe o cartelă nu trebuie să depășească valoarea 120, 72 și, respectiv, 80.

Dacă în instrucțiunea DISPLAY sînt citate date ale căror valori sînt reprezentate în virgulă mobilă, în cod complementar sau în zecimal-împachetat, transferul este precedat de conversia acestora în virgulă mobilă format extern sau zecimal-despachetat.

Dacă în instrucțiunea DISPLAY este citată o constantă figurativă, va fi transferat și înregistrat un singur caracter.

4) *Instrucțiunea* EXAMINE descrie operația de numărare a aparițiilor unui caracter (desemnat prin *literal-1/nume-dată-2*) într-un șir (desemnat prin *nume-dată-1*) și/sau înlocuirea acestui caracter printr-un alt caracter (desemnat prin *literal-2/nume-dată-3*).

Numărarea aparițiilor caracterului căutat are loc numai în cazul utilizării primului format al instrucțiunii; rezultatul este înregistrat (în cod complementar semicuvînt) într-un registru special, numit TALLY (rezervat de compilator în orice program COBOL), care poate fi citat în diviziunea de prelucrare.

Semnificația opțiunilor ALL, LEADING, UNTIL FIRST și FIRST este următoarea:

– ALL, indică numărarea și/sau înlocuirea tuturor aparițiilor caracterului căutat;

– LEADING, indică numărarea și/sau înlocuirea aparițiilor caracterului căutat, care sînt succesive și la începutul șirului;

– UNTIL FIRST, indică numărarea și/sau înlocuirea caracterelor care preced prima apariție a caracterului căutat;

– FIRST, indică înlocuirea primei apariții a caracterului căutat.

Observație. Numărarea și înlocuirea caracterului căutat se fac de la stînga la dreapta.

5) *Instrucțiunea* TRANSFORM descrie operația de cercetare și modificare a unui șir (desemnat prin *nume-dată*), conform unei anumite reguli, indicată de opțiunile citate în instrucțiune (tabelul 5.12).



Scanned with OKEN Scanner

*Tabelul 5.12*

**Instrucțiunea TRANSFORM**

| Opțiunea FROM | Opțiunea TO | Regula de transformare |
|---|---|---|
| *constantă-figurativă-1* | *constantă-figurativă-2*<br>*literal-2*<br>*nume-dată-2* | Caracterele șirului care sînt egale cu caracterul desemnat prin *constanta-figurativă-1* sînt înlocuite cu caracterul desemnat prin *constanta-figurativă-2/literal-2/nume-dată-2* |
| *literal-1*<br>*nume-dată-1* | *constantă-figurativă-2* | Caracterele șirului care sînt egale cu unul sau mai multe caractere ale datei *literal-1/nume-dată-1* sînt înlocuite cu caracterul desemnat prin *constanta-figurativă-2* |
| *literal-1*<br>*nume-dată-1* | *literal-2*<br>*nume-dată-2* | Dacă datele *literal-1/nume-dată-1* și *literal-2/nume-date-2* au același număr de caractere, caracterele șirului care sînt egale cu unul sau mai multe caractere ale datei *literal-1/nume-dată-1* sînt înlocuite cu caracterele datei *literal-2/nume-dată-2*<br>Dacă data *literal-2/nume-dată-2* conține un singur caracter, toate caracterele șirului care sînt egale cu acesta, vor fi înlocuite (cu acest caracter)<br>Utilizarea opțiunilor *nume-dată-1* și *nume-dată-2* în aceeași instrucțiune, impune ca datele ce le corespund să aibă același număr de caractere |

Observații. Datele *nume-dată-1* și *nume-dată-2* pot fi date grupate sau date elementare, alfabetice sau alfanumerice, iar lungimea zonelor ce le sînt rezervate nu trebuie să depășească 257 locații.

Literalele *literal-1* și *literal-2* trebuie să fie nenumerice. Valorile datelor *nume-dată-1/literal-1* nu pot conține caractere care se repetă.

5.1.9. **Prelucrarea fișierelor.** Limbajul COBOL dispune de clauze și instrucțiuni care permit descrierea fișierelor precum și a operațiilor pe aceste fișiere. Trei tipuri de organizare a fișierelor pot fi definite în acest limbaj: organizarea *secvențială* (continuă sau înlănțuită), organizarea *selectivă* și organizarea *secvențială-indexată*. Prelucrarea fișierelor este definită numai la nivel logic și se realizează prin intermediul următoarelor operații: *creare, consultare, actualizare, sortare.*

Elementele legate de prelucrarea fișierelor în limbajul COBOL sînt sintetizate în tabelul 5.13.



Scanned with OKEN Scanner

*Tabelul 5.13*

**Caracteristicile fişierelor COBOL**

| Organizare | Suport | Acces | Formatul articolelor | Operaţii | Numărul zonelor tampon |
|---|---|---|---|---|---|
| secvenţială | cartela | secvenţial | fix | creare<br>consultare | {1, ... 16} |
| | hîrtia de imprimantă | | fix variabil | creare | |
| | bandă magnetică | | fix variabil nedefinit | creare<br>consultare | |
| | disc magnetic | | fix variabil nedefinit | creare<br>consultare<br>actualizare<br>ştergeri<br>modificări<br>sortare | |
| selectivă | disc magnetic | direct | fix variabil | creare<br>consultare<br>actualizare<br>ştergeri<br>modificări<br>adăugări | {1, 2}<br>1<br><br><br>{1, 2} |
| secvenţială-indexată | disc magnetic | secvenţial | fix variabil | creare<br>consultare<br>actualizare<br>ştergeri<br>modificări | {1, 2, 3} |
| | | direct | | consultare<br>actualizare<br>ştergeri<br>modificări<br>adăugări | |

5.1.9.1. *Descrierea fişierelor* se realizează în două secţiuni : secţiunea INPUT-OUTPUT (în paragrafele FILE-CONTROL şi I-O-CONTROL) şi secţiunea FILE.

*În paragraful* FILE-CONTROL (tabelul 5.14) fiecărui fişier îi este asociată o rubrică SELECT care furnizează, prin intermediul unor clauze de descriere, informaţiile generale ce caracterizează fişierul. Singurele clauze obligatorii sînt SELECT şi ASSIGN ; celelalte clauze sînt opţionale şi pot fi citate în orice ordine.



Scanned with OKEN Scanner

*Tabelul 5.14*

**Paragraful FILE-CONTROL**

```
FILE-CONTROL.
 {SELECT nume-fişier
   ASSIGN TO {idex-1 / idex-2}
  [ , RESERVE {întreg-1 / NO} ALTERNATE [{AREA / AREAS}]]
  [ , ACCESS MODE IS {SEQUENTIAL / RANDOM}]
  [ , ORGANIZATION MODE IS {SEQUENTIAL / INDEXED / RANDOM}]
  [, ACTUAL KEY IS nume-dată-1]
  [, RECORD KEY IS nume-dată-2]
  [, SYMBOLIC KEY IS nume-dată-3]} . . . .
```

*În paragraful* I-O-CONTROL (tabelul 5.15) sînt precizate tehnicile speciale utilizate în prelucrarea fişierelor. Toate clauzele acestui paragraf sînt opţionale.

*Tabelul 5.15*

**Paragraful I-O-CONTROL**

```
I-O-CONTROL.
 [APPLY LOCATE MODE ON nume-fişier-1 [, nume-fişier-2] . . .]
[; RERUN [ON nume-fişier-3] EVERY {{END OF {REEL / UNIT}} / {întreg-1 RECORDS} / {întreg-2 CLOCK-UNITS}} OF nume-fişier-4]
[; SAME [{RECORD / SORT}] AREA FOR nume-fişier-5 [, nume-fişier-6] . . .]
[; MULTIPLE FILE TAPE CONTAINS nume-fişier-7 [ , nume-fişier-8] . . .] .
```

Clauzele de descriere ce pot apărea în paragrafele FILE-CONTROL şi I-O-CONTROL sînt prezentate în tabelul 5.16.

*Tabelul 5.16*

**Clauzele paragrafelor FILE-CONTROL şi I-O-CONTROL**

| Clauza | Indică : | Opţiuni | Semnificaţie |
|---|---|---|---|
| SELECT | numele fişierului | | |
| ASSIGN | identificatorul de exploatare (*idex*) | *idex-1* (SYSIN, SYSOUT, SYSPUNCH) | fişierului îi este afectată o unitate standard a sistemului (unitatea de intrare, ieşire, perforare) |
| | | *idex-2* (o literă urmată de două caractere) | fişierului îi este afectată o unitate periferică oarecare, cele două caractere (CR, CP, PR, MT, RD, MD, AD etc.) precizează tipul acesteia. |
| ORGANIZATION | organizarea fişierului | SEQUENTIAL (opţiune implicită) | organizare secvenţială |
| | | INDEXED | organizare secvenţială-indexată |
| | | DIRECT | organizare selectivă |
| ACCESS | modul de acces | SEQUENTIAL (opţiune implicită) | acces secvenţial |
| | | RANDOM | acces direct |
| RESERVE | numărul de zone -tampon | NO | fişierului îi este asociată o zonă-tampon |
| | | O | fişierului îi este asociată o zonă-tampon |
| | | *întreg-1* | fişierului îi este asociat un număr suplimentar de zone-tampon, egal cu *întreg-1* ($1 \leqslant$ *întreg-1* $\leqslant 14$); în absenţa acestei clauze, compilatorul asociază fişierului o zonă-tampon (fişiere selective) sau două zone-tampon (celelalte fişiere) |
| RERUN | crearea de puncte de reluare | END OF {REEL / UNIT} | la detectarea sfîrşitului unui volum al fişierului *nume-fişier-4* |
| | | *întreg-1* | după citirea/scrierea a *întreg-1* articole în/din fişierul *nume-fişier-4* |
| RECORD | data care reprezintă cheia de identificare a articolelor fişierelor secvenţiale-indexate | | |
| SYMBOLIC | — data care reprezintă cheia de identificare a articolelor fişierelor selective<br>— data (numită cheie simbolică) care reprezintă cheia pentru regăsirea, în acces direct, a articolelor fişierelor secvenţiale-indexate | | |
| ACTUAL | data (numită cheie actuală) ale cărei valori, stabilite prin algoritmul de randanizare, indică numărul căsuţelor unui fişier selectiv (cheia actuală trebuie descrisă prin clauzele PIC S9 (5) şi COMP) | | |
| APPLY | modul de acces la articole este LOCATE (acces în zona tampon) | | |
| MULTIPLE FILE | numele fişierelor care alcătuiesc o grupare (un multifişier) pe bandă sau disc | | |

Scanned with OKEN Scanner

Fiecărui fișier definit într-o rubrică SELECT trebuie să-i corespundă, în secțiunea FILE, o rubrică FD (File Description), în care este continuată descrierea fișierului. Rubrica FD trebuie să fie urmată de descrierea articolelor asociate fișierului. Singura clauză obligatorie a rubricii FD este clauza LABEL (tabelul 5.17).

*Tabelul 5.17*

**Rubrica de descriere a fișierelor**

FD *nume-fișier*

; LABEL { RECORD IS / RECORDS ARE } { OMITTED / STANDARD / *nume-dată-1* [*nume-dată-2*] ... }

[ ; RECORDING MODE IS < F / V / U > ]

[ ; RECORD CONTAINS *întreg-1* [TO *întreg-2*] CHARACTERS]

[ ; BLOCK CONTAINS *întreg-3* [TO *întreg-4*] { RECORDS / CHARACTERS } ]

[ ; DATA { RECORD IS / RECORDS ARE } *nume-articol-1* [, *nume-articol-2*] ... ]

[ ; { REPORT IS / REPORTS ARE } *nume-raport-1* [, *nume-raport-2*] ... ] .

În tabelul 5.18 este prezentată, succint, semnificația clauzelor rubricii FD.

5.1.9.2. *Instrucțiuni pentru prelucrarea fișierelor.* Descrierea operațiilor de acces la fișiere și la articole se realizează prin intermediul instrucțiunilor prezentate în tabelele 5.19 și 5.20.

În legătură cu aceste instrucțiuni se pot face observațiile prezentate în continuare.

1. Clauza ADVANCING este specifică fișierelor la imprimantă, precizînd condițiile în care va avea loc imprimarea rîndului. După cum este utilizată opțiunea BEFORE sau AFTER, imprimarea se va face înainte sau după avansarea hîrtiei cu un anumit număr de rînduri. Acest număr depinde de valoarea indicată prin *întreg* sau prin valoarea datei *nume-dată-2*, care poate fi o dată numerică de lungime egală cu unul sau două caractere sau o dată alfanumerică cu lungimea de un caracter.



Scanned with OKEN Scanner

*Tabelul 5.18*

**Clauzele rubricii FD**

| Clauza | Indică : | Opţiunea | Semnificaţia |
|---|---|---|---|
| RECORDING | formatul articolelor | F (opţiune implicită) | format fix |
| | | V | format variabil |
| | | U | format nedefinit |
| LABEL | prezenţa sau absenţa etichetelor de identificare a fişierului | OMITTED | Fişierului nu îi sînt asociat etichete (fişiere pe cartele, la imprimantă, pe bandă) |
| | | STANDARD | fişierului îi sînt asociate etichete standard (fişiere pe disc şi bandă) |
| | | *nume-data-1*... | fişierului îi sînt asociaţi etichete standard şi utilizator (fişier pe bandă) |
| DATA | numele articolelor | | |
| RECORD | lungimea zonei-articol | *întreg-1* | lungimea zonei-articol este egală cu *întreg-1* (cazul fişierelor cu articol de format fix) |
| | | *întreg-1* TO *întreg-2* | lungimea zonei-articol este egală cu *întreg-2* (cazul fişierelor cu articole de format variabil) |
| BLOCK | dimensiunea zonei-tampon | *întreg-3* | *întreg-1* indică numărul de locaţii (CHARACTERS) sau numărul de articole (RECORDS) conţinute de zona tampon (cazul fişierelor cu articole de format fix) |
| | | *întreg-3* TO *întreg-4* | *întreg-3* şi *întreg-4* indică numărul minim şi, respectiv, numărul maxim de locaţii (CHARACTERS) sau numărul minim şi, respectiv, maxim de articole (RECORDS) ce pot fi conţinute de zona tampon (cazul fişierelor care nu au articole de format fix) |
| REPORT | numele rapoartelor care utilizează suportul afectat fişierului | | |

Scanned with OKEN Scanner

*Tabelul 5.19*

**Instrucţiuni pentru prelucrarea fişierelor**

OPEN

```
                 INPUT {nume-fişier [{REVERSED                  }]} ...
OPEN  {          OUTPUT {nume-fişier [WITH NO REWIND]} ...        } ...
                 {INPUT-OUTPUT}
                 {I-O          } {nume-fişier} ...
```

CLOSE

```
CLOSE { nume-fişier [{ REEL }] [ WITH { NO REWIND }]} ...
                     { UNIT }          { LOCK      }
```

READ

**Format 1:**

READ *nume-fişier* [RECORD] [INTO *nume-dată*] ; AT END *instrucţiune* ....

**Format 2:**

READ *nume-fişier* [RECORD] [INTO *nume-dată*] ; INVALID KEY *instrucţiune* ....

WRITE

**Format 1:**

```
WRITE nume-articol [FROM nume-dată-1]  [{BEFORE} ADVANCING
                                        {AFTER }
                                                {nume-dată-2} LINES ]
                                                {întreg     }
```

**Format 2:**

WRITE *nume-articol* [FROM *nume-dată*] ; INVALID REY *instrucţiune* ....

DELETE

DELETE *nume-articol*

REWRITE

REWRITE *nume-articol* [FROM *nume-dată*] ; INVALID KEY *instrucţiune* ....

*Tabelul 5.20*

**Instrucțiunile pentru prelucrarea fișierelor**

| Instrucțiune | Format | Descrie operația de : | Opțiuni/clauze | Semnificație |
|---|---|---|---|---|
| OPEN | | deschidere a unuia sau mai multor fișiere | INPUT | fișierul este consultat |
| | | | OUTPUT | fișierul este creat |
| | | | I-O / INPUT-OUTPUT | fișierul pe disc este actualizat |
| | | | REVERSED | fișierul pe bandă este consultat de la ultimul articol către primul |
| | | | NO REWIND | suprimarea rebobinării benzii, care însoțește în mod normal deschiderea fișierului |
| CLOSE | | închidere a unuia sau mai multor fișiere | REEL / UNIT | fișierul ocupă mai multe volume bandă sau disc |
| | | | NO REWIND | suprimarea rebobinării benzii, care însoțește în mod normal închiderea fișierului |
| READ | 1 | citire a unui articol dintr-un fișier secvențial | fără clauza INTO | articolul citit este transferat în zona-articol |
| | | | cu clauza INTO | articolul citit este transferat în zona-articol și în zona *nume-dată* |
| | | | AT END | indică instrucțiunile ce vor fi executate la detectarea sfîrșitului fișierului |
| | 2 | citire, în acces direct, a unui articol dintr-un fișier selectiv sau secvențial-indexat | fără clauza INTO | articolul citit este transferat în zona-articol |
| | | | cu clauza INTO | articolul citit este transferat în zona-articol și în zona *nume-dată* |
| | | | INVALID KEY | indică instrucțiunile ce vor fi executate dacă : a) în fișierul secvențional-indexat nu există un articol care să aibă o valoare a cheii egală cu valoarea cheii simbolice ; b) în fișierul selectiv, în căsuța indicată de valoarea cheii actuale, nu există un articol care să aibă o valoare a cheii actuale egală cu valoarea cheii simbolice, sau valoarea cheii actuale nu aparține intervalului $(0, m-1)$ |

Scanned with OKEN Scanner

*Tabelul 5.20* (continuare)

| | | | | |
|---|---|---|---|---|
| WRITE | 1 | Scrierea unui articol într-un fişier secvenţial | fără clauza FROM | articolul este constituit, pentru scriere, în zona-articol |
| | | | cu clauza FROM | articolul este constituit, pentru scriere, în zona *nume-dată* |
| | | | ADVANCING | clauza este specifică fişierelor la imprimantă, precizînd condiţiile de imprimare a articolului |
| | 2 | Scrierea unui articol într-un fişier selectiv sau secvenţial-indexat | fără clauza FROM | articolul este constituit, pentru scriere, în zona *nume-dată* |
| | | | cu clauza FROM | articolul este constituit, pentru scriere, în zona *nume-dată* |
| | | | INVALID KEY | indică instrucţiunile ce vor fi executate dacă : a) valoarea cheii de identificare a articolului nu este strict mai mare decît cea a articolului scris anterior (operaţia de creare), sau în fişier există deja un articol cu aceeaşi valoarea a cheii (operaţia de adăugare) — în cazul fişierelor secvenţiale-indexate ; b) valoarea cheii de identificare a articolului coincide cu cea a unui articol din căsuţa indicată prin valoarea cheii actuale, sau valoarea cheii actuale nu aparţine intervalului (0, *m*—1) |
| DELETE | | Ştergerea unui articol citit anterior dintr-un fişier pe disc | | în cazul fişierelor secvenţiale şi secvenţiale-indexate ştergerea este logică ; în cazul fişierelor selective ştergerea este fizică, dacă articolul se află în zona principală, şi logică, dacă articolul se află în zona de depăşire |
| REWRITE | | rescrierea într-un fişier pe disc a unui articol citit anterior, ale cărui date au fost modificate total sau parţial | fără clauza FROM | articolul de rescris se află în zona-articol |
| | | | cu clauza FROM | articolul de rescris se află în zona *nume-dată* |
| | | | INVALID KEY | indică instrucţiunile ce vor fi executate dacă modificarea datelor articolului fişierului secvenţial-indexat a afectat valoarea cheii de identificare |



Scanned with OKEN Scanner

Semnificaţia valorilor indicate prin *întreg* sînt :

0 — hîrtie avansează pînă la începutul unei noi pagini ;
1 — hîrtia avansează cu un rînd ;
— — — —

Opțiunea *întreg-2* TO *întreg-3* TIMES este utilizată pentru definirea tablourilor cu număr variabil de elemente ; *întreg-2* (a cărui valoare poate fi zero) indică numărul minim, *întreg-3* numărul maxim, iar valoarea datei *nume-dată-3* (care nu trebuie să depășească valoarea lui *întreg-3*) indică numărul exact de elemente într-un anumit moment al prelucrării.

Opțiunile ASCENDING sau DESCENDING indică faptul că elementele tabloului sînt ordonate crescător (ASCENDING) sau descrescător (DESCENDING), în funcție de valorile cheilor *nume-dată-4*, *nume-dată-5*, ... Aceaste chei trebuie să fie definite în rubrica în care apare clauza OCCURS sau într-o rubrică subordonată acesteia. În cazul în care există mai multe chei, prioritatea acestora este indicată în ordine descrescătoare.

Clauza OCCURS poate fi citată numai în rubricile de descriere a datelor al căror număr de nivel aparține mulțimii {02, ..., 49}. Prima formă a clauzei poate fi citată în maximum trei rubrici de descriere succesive (ale căror numere de nivel, notate cu *n1*, *n2* și *n3* satisfac relația $n1<n2<n3$), permițînd definirea tablourilor cu maximum trei dimensiuni.

A doua formă a clauzei poate apărea într-o singură rubrică și anume în cea care descrie datele (elementare sau grupate) ultimului nivel al unui articol.

Privită în corelație cu clauzele VALUE și REDEFINES, utilizarea clauzei OCCURS are două restricții :

— elementele unui tablou nu pot fi inițializate ;

— un tablou cu număr variabil de elemente nu poate fi redefinit.

5.1.10.1. *Identificarea elementelor tablourilor.* Individualizarea elementelor unui tablou se poate realiza utilizînd indici sau indecși.

Un *indice* poate fi un literal numeric, o dată numerică elementară sau registrul special TALLY. Valoarea unui indice trebuie să reprezinte un număr întreg mai mare decît zero și mai mic decît numărul maxim de elemente ale tabloului (număr definit prin clauza OCCURS). Un indice sau un ansamblu de indici care identifică un element al unui tablou trebuie scris între paranteze, imediat după spațiul care urmează numelui elementului. Numele elementului este cel atribuit în rubrica asociată ultimei dimensiuni a tabloului sau într-o rubrică subordonată acesteia. Dacă identificarea elementului se realizează prin mai mulți indici (cazul tablourilor multidimensionale), aceștia trebuie separați prin cel puțin un spațiu sau prin virgulă urmată de cel puțin un spațiu.

Un *index* reprezintă un tip aparte de indice, utilizat exclusiv pentru identificarea elementelor tablourilor. Spre deosebire de un indice obișnuit, un index reprezintă adresa relativă în cadrul tabloului a zonei de memorie rezervată unui element. Utilizarea indecșilor este avantajoasă, întrucît simplifică adresarea elementelor unui tablou.



Scanned with OKEN Scanner

Definirea unui index se realizează prin opțiunea INDEXED BY a clauzei OCCURS. Pentru memorarea valorilor unui index compilatorul rezervă o zonă de memorie de patru locații.

Formatul opțiunii INDEXED indică faptul că unul sau mai mulți indecși pot fi asociați fiecărei dimensiuni a unui tablou. Această asociere trebuie respectată atunci cînd sînt adresate elementele tabloului, adică un index asociat unei dimensiuni a tabloului poate fi utilizat numai pentru identificarea elementelor după acea dimensiune.

Există două modalități de adresare prin utilizarea indecșilor :

– *directă*, cînd se indică indexul, de exemplu : B (I, J) ; B (I, K) ; GRUP (L) ; CANTITATE (L) ;

– *relativă*, cînd se indică indexul urmat de unul din operatorii + sau – și de un literal numeric ce desemnează un număr întreg, de exemplu :

B (I+2, J–1) ; CANTITATE (L+3).

Valoarea unui index poate fi stabilită numai prin instrucțiunile SET, SEARCH sau PERFORM.

5.1.10.2. *Date-index*. Prelucrarea tablourilor necesită, uneori, un tip aparte de date, numite *date-index*, ale căror valori au aceeași semnificație ca și valorile indecșilor.

O dată-index este definită prin opțiunea INDEX a clauzei USAGE. Întrucît datelor-index compilatorul le rezervă zone de memorie de patru locații, în care valorile sînt reprezentate sub formă de cuvînt, în rubricile de descriere ce le sînt asociate nu are sens utilizarea clauzelor PICTURE, SYNCRONIZED, JUSTIFIED, VALUE, BLANK WHEN ZERO. Datelor-index le sînt atribuite valori prin instrucțiunile SET și SEARCH.

1) *Instrucțiunea* SET descrie operațiile de atribuire de valori indecșilor, datelor-index sau datelor elementare numerice întregi.

În cazul primului format al instrucțiunii (tabelul 5.21) valoarea ce se atribuie este desemnată prin una din datele menționate după cuvîntul

*Tabelul 5.21*

**Instrucțiunile SET și SEARCH**

**SET**

Format 1 :

SET { *nume-index-1* / *nume-dată-1* / *nume-dată-index-1* } [ , { *nume-index-2* / *nume-dată-2* / *nume-dată-index-2* } ] ... TO { *nume-index-3* / *nume-dată-3* / *nume-dată-index-3* / *literal-1* }

Format 2 :

SET *nume-index-4* [, *nume-index-5*] ... { UP BY / DOWN BY } { *nume-dată-4* / *literal-2* }



Scanned with OKEN Scanner

SEARCH

Format 1:

SEARCH *nume-dată-1* [VARYING {*nume-index* / *nume-dată-2*}]

[; AT END *instrucțiune-1* ...]

; WHEN *condiție-1* {*instrucțiune-2* ... / NEXT SENTENCE}

[; WHEN *condiție-2* {*instrucțiune-3* ... / NEXT SENTENCE}]

Format 2:

SEARCH ALL *nume-dată-1*

[; AT END *instrucțiune-1* ...]

; WHEN *condiție-1* {*instrucțiune-2* ... / NEXT SENTENCE}

TO, iar data (datele) ce primesc această valoare sînt menționate înaintea cuvîntului TO. Formele sintactice permise ale acestei forme a instrucțiunii SET sînt indicate în tabelul 5.22 prin litera P, iar cele nepermise, prin litera N.

*Tabelul 5.22*

**Instrucțiunea SET**

| Data receptoare / Data emițătoare | nume-index-3 | {nume-dată-index-3} | nume-dată-3 | literal-1 |
|---|---|---|---|---|
| *nume-index-1* ... | P | P | P | P |
| *nume-dată-index-1*... | P | P | N | N |
| *nume-dată-1* ... | P | N | N | N |

Formatul 2 al instrucțiunii SET descrie operația de mărire (UP BY) sau micșorarea (DOWN BY) a valorilor indecșilor *nume-index-4*, *nume-index-5*, ... cu valoarea literalului *literal-2* (care trebuie să fie un număr întreg pozitiv) sau cu valoarea datei *nume-data-4* (care nu trebuie să fie descrisă cu opțiunea INDEX).

2) *Instrucțiunea* SEARCH descrie operația de cercetare a elementelor unui tablou în scopul identificării unui element care îndeplinește o anumită condiție. Avantajul utilizării acestei instrucțiuni constă în faptul că pro-



Scanned with OKEN Scanner

gramatorul nu trebuie să descrie explicit un ciclu pentru a realiza această cercetare, ci trebuie să indice numai : numele tabloului ; numele datei căreia, la găsirea elementului căutat, i se va atribui ca valoare adresa relativă a elementului în cadrul tabloului ; condiţia sau condiţiile ce trebuie testate ; secvenţele de instrucţiuni ce vor fi executate la găsirea elementului căutat sau la terminarea explorării întregului tablou.

Instrucţiunea SEARCH are două formate (tabelul 5.21) ; formatul 1 este utilizat pentru căutarea secvenţială a elementelor tabloului, iar formatul 2 pentru căutarea nesecvenţială.

Indiferent de formatul utilizat, informaţiile furnizate de programator au următoarea semnificaţie :

— *nume-dată-1* indică numele tabloului ; descrierea sa trebuie să conţină clauza OCCURS şi opţiunea INDEXED ;

— *instrucţiune-1* indică secvenţa de instrucţiuni care va fi executată la terminarea explorării întregului tablou (deci în cazul în care nu conţine elementul căutat) ; în cazul în care clauza AT END lipseşte, execuţia programului continuă cu instrucţiunea ce urmează instrucţiunii SEARCH ;

— *condiţie-1*, *condiţie-2*, ... indică testele efectuate asupra valorilor datelor ce aparţin elementelor tabloului ;

— *instrucţiune-2*, *instrucţiune-3*, ... sau opţiunea NEXT SENTENCE indică secvenţele de instrucţiuni care vor fi executate la găsirea elementului ce îndeplineşte una dintre condiţiile stabilite.

În cazul formatului 1, condiţiile pot fi de orice tip şi sînt testate în ordinea în care apar în instrucţiune. Adresa relativă în cadrul tabloului a elementului care îndeplineşte una din condiţiile specificate va fi memorată în zona rezervată datei *nume-index* (definită prin opţiunea INDEXED în aceeaşi rubrică în care apare şi numele tabloului sau într-o rubrică ce descrie un alt tablou) sau *nume-data-2* (care poate fi o dată numerică întreagă elementară sau o dată-index definită prin clauza USAGE INDEX). De asemenea, *nume-index* sau *nume-dată-2* sînt utilizate şi ca variabile de control pentru cercetarea elementelor tabloului. În cazul în care opţiunea YARYING lipseşte, ca variabilă de control va fi utilizat primul index citat în opţiunea INDEXED. Înainte de execuţia instrucţiunii, variabilei de control trebuie să i se atribuie o valoare între 1 şi numărul maxim al elementelor tabloului ; în caz contrar, explorarea tabloului este abandonată.

În cazul formatului 2 între elementele tabloului trebuie să existe o relaţie de ordine, care este descrisă prin opţiunea ASCENDING sau DESCENDING. Condiţia poate fi simplă, de tip test de relaţie (EQUAL TO) sau test de nume de condiţie, sau compusă, formată prin reunirea condiţiilor simple, cu ajutorul operatorului logic AND. Testarea poate fi efectuată numai asupra valorilor cheilor care definesc relaţia de ordine a elementelor tabloului sau asupra valorilor datei, căreia îi este asociat



Scanned with OKEN Scanner

numele de condiție. În cercetarea elementelor tabloului, valoarea inițială a variabilei de control (care este primul index citat în opțiunea INDEXED) este ignorată.

5.1.10.3. *Prelucrarea tablourilor prin instrucțiunea* PERFORM. Definirea ciclurilor pentru prelucrarea elementelor tablourilor poate fi realizată comod în limbajul COBOL prin utilizarea formatului 4 al instrucțiunii PERFORM (vezi tabelul 5.10). Aceasta constă în precizarea :

– secvenței de prelucrare care formează corpul ciclului – prin *nume prelucrare-1* THRU *nume-prelucrare-2* ;

– indicelui/indexului asociat fiecărei dimensiuni a tabloului – prin clauza VARYING ;

– valori inițiale a fiecărui indice/index – prin clauza FROM ;

– valorii cu care este mărită, după fiecare pas al ciclului, valoarea fiecărui indice/index – prin clauza BY ;

– condiției care trebuie testată pentru stabilirea sfîrșitului execuției secvenței de prelucrare sau pentru execuția operațiilor necesare trecerii la următorul pas al unui ciclu – prin clauza UNTIL.

În funcție de numărul indicilor/indecșilor citați într-o instrucțiune PERFORM, aceasta poate descrie unul, două sau trei cicluri ; ultimele două reprezintă cicluri incluse unul în altul.

Condițiile citate în instrucțiunea PERFORM pot fi de tipul celor prezentate la instrucțiunea IF.

Valorile identificatorilor care apar în instrucțiunea PERFORM (*nume-dată*, *nume-index*, *literal*) trebuie să reprezinte numere întregi.

5.1.11. **Sortarea fișierelor.** Efectuarea operației de sortare implică, în general, prezența a trei fișiere :

– un *fișier de intrare*, care conține articolele ce trebuie să fie ordonate ;

– un *fișier de ieșire*, care va conține după efectuarea sortării – articolele fișierului de intrare ordonate conform cheilor de sortare stabilite ;

– un *fișier de manevră*, care este utilizat ca zonă de lucru în timpul sortării.

Organizarea acestor fișiere trebuie să fie secvențială, suportul de înregistrare poate fi banda sau discul magnetic, iar articolele pot fi de format fix sau variabil, grupate sau nu în blocuri.

Din punct de vedere funcțional, se poate considera că sortarea se realizează în trei etape :

– *etapa de constituire a fișierului de manevră*, în care articolele fișierului de intrare sînt grupate în șiruri monotone (crescătoare sau descrescătoare), numite monotonii ;



Scanned with OKEN Scanner

— *etapa de sortare propriu-zisă*, în care monotoniile sînt înregistrate în fișierul de manevră și apoi interclasate ;

— *etapa de exploatare a fișierului de manevră*, în care articolele acestuia sînt înregistrate pe suportul fișierului de ieșire.

În etapa de sortare propriu-zisă pot fi utilizate diverse metode ; modulul de sortare COBOL utilizează, în funcție de informațiile ce-i sînt transmise, una dintre metodele : *oscilantă*, *polifazelor*, *balanței* [119].

În general, sistemele de operare dispun de programe specializate în realizarea operațiilor de sortare. Avantajul utilizării modulului de sortare COBOL constă în faptul că permite inserarea unor proceduri speciale în prima și/sau ultima etapă a operației de sortare. Aceste proceduri sînt necesare atunci cînd articolele ce trebuie sortate nu aparțin unui fișier unic (de intrare) sau cînd articolele sortate nu trebuie grupate într-un fișier unic (de ieșire).

5.1.11.1. *Descrierea operației de sortare.* Operația de sortare este descrisă în limbajul COBOL prin intermediul unor rubrici, clauze și instrucțiuni specifice. Descrierea fișierului de manevră se realizează într-un mod particular ; celelalte fișiere care intervin în prelucrare se descriu în mod obișnuit.

În paragraful FILE-CONTROL, fișierului de manevră trebuie să-i corespundă o rubrică SELECT în care se indică : numele fișierului - prin clauza SELECT ; identificatorul de exploatare și tipul unității periferice — prin clauza ASSIGN. În funcție de tipul perifericului indicat și prezența sau absența opțiunii BSO în ordinul COMPILE, în etapa a doua a sortării va fi utilizată ;

— metoda oscilantă, dacă tipul unității periferice este discul magnetic ;

— metoda polifazelor, dacă tipul unității periferice este banda magnetică ;

— metoda balanței, dacă tipul unității periferice este discul magnetic, iar în cartela de comandă COMPILE figurează opțiunea BSO.

În secțiunea FILE, descrierea fișierului de manevra este continuată prin rubrica SD în care se indică numele fișierului și numele articolului (articolelor). Rubrica SD trebuie să fie urmată de descrierea articolului (articolelor) fișierului de manevră, care trebuie să pună în evidență cheile de sortare și, eventual, datele asupra cărora se operează prin prelucrările speciale introduse în prima și/sau ultima etapă a sortării.

5.1.11.2. *Instrucțiuni de sortare.* Există trei instrucțiuni specifice sortării : SORT, RELEASE și RETURN (tabelul 5.23).

Execuția instrucțiunii SORT realizează apelul modulului de sortare, căreia îi transmite informații referitoare la : fișierele care intervin în pre-



Scanned with OKEN Scanner

*Tabelul 5.23*

**Instrucţiuni pentru sortarea fişierelor**

SORT

SORT *fişier-manevră* {ON {ASCENDING / DESCENDING} KEY *nume-dată-1* [, *nume-dată-2*] ... }...

{INPUT PROCEDURE IS / USING *fişier-intrare*} *nume-secţiune-1* [THRU *nume-secţiune-2*]

{OUTPUT PROCEDURE IS / GIVING *fişier-ieşire*} *nume-secţiune-3* [THRU *nume-secţiune-4*]

RELEASE

RELEASE *nume-articol* [FROM *nume-dată*]

RETURN

RETURN *fişier-manevră* [RECORD] [INTO *nume-dată*] ; AT END *instrucţiune* ....

lucrare, cheile de sortare, sensul sortării şi procedurile speciale ce vor fi, eventual, executate în prima şi/sau ultima etapă a sortării.

Cheile de sortare, al căror număr maxim este 12, iar suma lungimilor zonelor ce le sînt rezervate nu trebuie să depăşească 256 locaţii, sînt indicate prin *nume-dată-1*, *nume-dată-2*, ... care desemnează date ale articolelor fişierului de manevră. Tipul acestor date, reprezentarea internă, dimensiunea şi poziţia lor în cadrul articolelor sînt precizate prin clauzele din rubricile în care sînt descrise.

Prioritatea cheilor este dată de ordinea în care acestea sînt citate în instrucţiunea SORT, adică prima cheie are prioritatea cea mai mare, iar ultima, prioritatea cea mai mică.

Sensul sortării este definit, pentru fiecare cheie sau pentru un grup de chei, prin opţiunile ASCENDING (crescător) şi DESCENDING (descrescător).

În cazul în care articolele, ce vor fi supuse sortării, şi/sau cele rezultate prin sortare, sînt grupate într-un fişier unic, poate fi utilizată opţiunea USING (care specifică fişierul de intrare) şi/sau opţiunea GIVING (care specifică fişierul de ieşire). În caz contrar, pot fi utilizate opţiunile INPUT PROCEDURE (procedura de intrare) şi/sau OUTPUT PROCEDURE (procedura de ieşire). Introducerea procedurilor speciale implică structurarea diviziunii PROCEDURE în secţiuni.

Procedura de intrare/ieşire grupează, în cadrul uneia sau mai multor secţiuni consecutive (desemnate, prima prin *nume-secţiune-1*/*nume-secţiune-3*, iar ultima prin *nume-secţiune-2*/*nume-secţiune-4*) instrucţiunile care vor realiza constituirea/exploatarea fişierului de manevră.

Procedura de intrare trebuie să conţină cel puţin o instrucţiune RELEASE, iar procedura de ieşire, cel puţin o instrucţiune RETURN. Instrucţiunile RELEASE şi RETURN sînt asemănătoare instrucţiunilor WRITE şi READ pentru acces secvenţial, realizînd scrierea/citirea unui articol în/din fişierul de manevră.

În legătură cu utilizarea instrucţiunii SORT pot fi făcute următoarele precizări ;

— în instrucţiunea SORT poate fi combinată opţiunea USING/GIVING cu opţiunea INPUT PROCEDURE/OUTPUT PROCEDURE ;

— procedura de intrare nu trebuie să conţină secţiuni comune cu procedura de ieşire ;

— într-o procedură de intrare/ieşire nu trebuie să figureze instrucţiunea SORT, iar instrucţiunile GO TO sau PERFORM — dacă sînt utilizate — nu trebuie să facă referiri la secţiuni/paragrafe exterioare acestei proceduri ;

— secţiunile diviziunii PROCEDURE care nu aparţin unei proceduri de intrare/ieşire nu trebuie să conţină instrucţiuni GO TO sau PERFORM care fac referiri la secţiuni/paragrafe ale acestei proceduri ;

— fişierul de manevră şi fişierele citate în opţiunile USING şi GIVING nu trebuie să fie menţionate în instrucţiunile OPEN şi CLOSE.

### 5.1.12. Editarea rapoartelor

5.1.12.1. *Structura rapoartelor.* Realizarea unui raport necesită prezenţa unui fişier de intrare ale cărui articole sînt ordonate, crescător sau descrescător, după valorile unuia sau mai multor date, numite *variabile de control.* Articolele consecutive ale fişierului pentru care valorile unei variabile de control sînt egale, alcătuiesc o grupă, numită *grupă de control.*

În cadrul raportului, fiecărui articol şi/sau grupe de control îi pot fi asociate una sau mai multe linii, care alcătuiesc aşa-numitele *grupe de editare.* Grupa de editare asociată unui articol se numeşte grupă de tip DETAIL şi conţine date (care au fost supuse sau nu unor operaţii de prelucrare) provenite, în general, din articolele fişierului de intrare. Unei grupe de control îi pot fi asociate două grupe de editare numite : CONTROL HEADING — tipărită înaintea unei grupe DETAIL a unei grupe de control şi care conţine, în general, elemente de identificare a acesteia ; CONTROL FOOTING — tipărită după ultima grupă DETAIL a unei grupe de control şi care conţine, în general, totaluri ale valorilor datelor ce aparţin articolelor unei grupe de control.

În cazul în care raportul conţine mai multe pagini, fiecărei pagini îi pot fi asociate două tipuri de grupe de editare numite : PAGE HEADING — tipărită la începutul paginii şi care conţine, în general, elementele



Scanned with OKEN Scanner

de identificare a acesteia ; PAGE FOOTING – tipărită la sfîrşitul paginii şi care conţine, de obicei, totaluri de pagină.

Grupele de editare care conţin elementele de identificare ale unui raport se numesc grupe de tip REPORT HEADING, iar cele care marchează sfîrşitul raportului şi conţin totaluri generale se numesc grupe de tip REPORT FOOTING.

Prezenţa sau absenţa unui anumit tip de grupă de editare în cadrul unui raport este în funcţie de formatul raportului şi opţiunile utilizatorului.

Pentru organizarea, în cadrul rapoartelor, a grupelor de editare şi a liniilor care le compun, editorul de rapoarte generează, pentru fiecare raport, două registre speciale, identificate prin cuvinte cheie : PAGE-COUNTER (contor de pagini) şi LINE-COUNTER (contor de linii) ; aceste registre pot fi referite în program. În cazul în care în program sînt definite mai multe rapoarte, referirea impune şi citarea (prin calificare) numelui ce se atribuie raportului.

5.1.12.2. *Descrierea rapoartelor.* Liniile ce alcătuiesc un raport reprezintă articole ale unui fişier, definit în program ca fişier de ieşire. În general, fişierul ce conţine raportul are ca suport hîrtia de imprimat ; este posibilă (atunci cînd raportul este editat ulterior) şi utilizarea altui suport.

Acestui fişier trebuie să-i corespundă o descriere obişnuită în paragraful FILE-CONTROL. Rubrica FD, care-i este asociată în secţiunea FILE, trebuie să indice numele raportului (prin clauza REPORT) şi, eventual, lungimea maximă a articolelor (prin clauza RECORD), ţinînd seama de codul de salt. Rubrica FD nu trebuie să fie urmată de descrierea articolelor asociate fişierului de ieşire.

*Rubrica* RD. În secţiunea REPORT este definită, prin intermediul rubricii RD (tabelul 5.24), structura fizică şi organizarea raportului. Numele raportului indicat în această rubrică trebuie să coincidă cu cel indicat în rubrica FD.

*Tabelul 5.24*

**Rubrica de descriere a rapoartelor**

RD nume-raport [; WITH CODE *literal*]

[ ; { CONTROL LS / CONTROLS ARE } { FINAL / *nume-dată-1, nume-dată-2 ...* / FINAL *nume-dată-1, nume-dată-2 ...* } ]

[ ; PAGE [ { LIMIT IS / LIMITS ARE } ] *întreg-1* { LINE / LINES }

[, HEADING *întreg-2*]

[, FIRST DETAIL *întreg-3*

[, LAST DETAIL *întreg-4*]

[, FOOTING *întreg-5*]].

*Clauza* CODE specifică, prin *literal*, elementul de identificare ce se asociază fiecărei linii a raportului ; literalul, a cărui lungime este de două caractere, trebuie să fie nenumeric. Utilizarea acestei clauze permite distingerea liniilor unui raport, în cazul în care fișierul de ieșire conține mai multe rapoarte.

*Clauza* CONTROL indică variabilele de control (*FINAL*, *nume-dată-1*, *nume-dată-2*, ...) și prioritatea (în ordine descrescătoare) efectuării controalelor. Controalele constau în testarea (înaintea generării unei grupe de tip DETAIL) valorilor variabilelor de control și se efectuează începînd cu variabila cu prioritatea cea mai mare. Dacă testarea indică schimbarea valorii unei variabile, imprimarea grupei de tip DETAIL este suspendată temporar pentru : imprimarea grupelor de tip CONTROL FOOTING ce corespund valorilor variabilei testate și ale variabilelor care au prioritate mai mică decît variabila testată, imprimarea grupelor de tip CONTROL HEADING ce corespund noilor valori ale variabilei testate și ale variabilelor care au prioritate mai mică decît variabila testată.

*Opțiunea* FINAL indică nivelul de control maxim, acționînd la începutul și la sfîrșitul raportului, adică înaintea imprimării primei linii DETAIL și după imprimarea ultimei linii DETAIL.

*Clauza* PAGE LIMIT precizează dimensiunea și structura unei pagini a raportului :

– *întreg-1* indică numărul maxim de linii ce vor fi imprimate pe o pagină ;

– *întreg-2* indică poziția (în cadrul paginii) primei linii a unei grupe de tip REPORT HEADING sau de tip PAGE HEADING :

– *întreg-3* indică poziția primei linii a unei grupe de DETAIL, CONTROL HEADING sau CONTROL FOOTING ;

– *întreg-4* indică poziția ultimei linii a unei grupe DETAIL sau CONTROL HEADING ;

– *întreg-5* indică poziția ultimei linii a unei grupe CONTROL FOOTING.

Evident, $1 \leqslant$ *întreg-2* $\leqslant$ *întreg-3* $\leqslant$ *întreg-4* $\leqslant$ *întreg-5* $\leqslant$ *întreg-1* $\leqslant 999$.

Rubrica RD este urmată de rubricile de descriere a grupelor de editare și a liniilor care compun aceste grupe (tabelul 5.25). Formatul 1 este utilizat pentru definirea grupelor de editare, iar formatul 2 pentru definirea liniilor precum și a datelor ce le compun.

*Rubrici de descriere a grupelor de editare. Clauza* TYPE indică tipul grupelor de editare și momentul în care acestea vor fi imprimate și anume :

– la începutul raportului (REPORT HEADING) ;

– la începutul paginii (PAGE HEADING) ;



Scanned with OKEN Scanner

*Tabelul 5.25*

**Rubrica de descriere a grupelor şi liniilor rapoartelor**

Format 1:

```
01 [nume-dată-1] [; USAGE IS DISPLAY]

                 { {REPORT HEADING} }
                 { {RH              }
                 { {PAGE HEADING}
                 { {PH           }
                 { {CONTROL HEADING} {nume-dată-2}
                 { {CH              } {FINAL      }
; TYPE IS        { {DETAIL}
                 { {DE    }
                 { {CONTROL FOOTING} {nume-dată-3}
                 { {CF              } {FINAL      }
                 { {PAGE FOOTING}
                 { {PF           }
                 { {REPORT FOOTING}
                 { {RF             }

[ ; LINE NUMBER IS {intreg-1 [ON NEXT PAGE]}
                   {ON NEXT PAGE            }
                   {PLUS intreg-2           }]

[ ; NEXT GROUP IS {intreg-3          }
                  {PLUS intreg-4     }
                  {ON NEXT PAGE      }]
```

Format 2:

```
număr-nivel [nume-dată-1] [; BLANK WHEN ZERO]

[ ; LINE NUMBER IS {intreg-1 ON NEXT PAGE}
                   {ON NEXT PAGE          }
                   {PLUS intreg-2         }]

; GROUP INDICATE]

[; {JUSTIFIED} RIGHT]
   {JUST     }

[; COLUMN NUMBER IS intreg-3]

[ ; {PICTURE} IS şablon]
   {PIC     }
```

[; RESET ON { *nume-dată-2* / FINAL }]

[; SOURCE IS *identificator-1*]

[; VALUE IS *literal*]

[; USAGE IS DISPLAY]

[; SUM *identificator-2* [, *identificator-3*] ... [UPON *nume-dată-3* ...]].

– la începutul primei grupe de control, în cazul opțiunii FINAL, sau la începutul fiecărei grupe de control stabilită prin testarea variabilei de control *nume-dată-2* (CONTROL HEADING);

– după citirea unui articol din fișierul de intrare (DETAIL);

– la sfîrșitul ultimei grupe de control, în cazul opțiunii FINAL, sau la sfîrșitul fiecărei grupe de control stabilite prin testarea variabilei *nume-dată-3* (CONTROL FOOTING);

– la sfîrșitul paginii (PAGE FOOTING);

– la sfîrșitul raportului (REPORT FOOTING).

Grupele de editare de tip REPORT HEADING, PAGE HEADING, PAGE FOOTING și REPORT FOOTING nu pot fi definite decît o singură dată în cadrul unui raport. Utilizarea opțiunilor *FINAL*, *nume-dată-2* sau *nume-dată-3* implică citarea lor și în clauza CONTROL a rubricii RD.

*Clauza* LINE precizează pagina și poziția din cadrul paginii în care va fi imprimată linia ce aparține grupei de editare în rubrica căreia apare această clauză. Semnificația opțiunilor clauzei este:

– *întreg-1*: linia va fi imprimată pe pagina curentă, în poziția indicată de *întreg-1* ($1 \leqslant$ *întreg-1* $\leqslant 999$);

– *întreg-1* ON NEXT PAGE: linia va fi imprimată pe pagina următoare, în poziția indicată de *întreg-1*;

– ON NEXT PAGE: linia va fi imprimată pe pagina următoare, în poziția indicată prin clauza PAGE LIMIT;

– PLUS *întreg-2*: linia va fi imprimată pe pagina curentă, în poziția relativă (față de linia tipărită anterior) indicată de *întreg-2* ($2 \leqslant$ *întreg-2* $\leqslant 999$).

Clauza LINE trebuie să apară în fiecare rubrică ce descrie o linie a unei grupe de editare, iar opțiunea NEXT PAGE trebuie să apară numai în rubrica ce descrie prima linie a grupei.

*Clauza* NEXT GROUP precizează numărul de linii lăsate după scrierea ultimei linii a grupei de editare descrisă prin rubrica în care apare această clauză. În funcție de opțiunea specificată, valorile contorului de linii și contorului de pagini (stabilite după scrierea ultimei linii) vor fi:



Scanned with OKEN Scanner

– *întreg-3* : contorului de linii i se atribuie valoarea lui *întreg-3* ;
– PLUS *întreg-4* : valoarea contorului de linii este mărită cu valoarea lui *întreg-4* ;
– ON NEXT PAGE : contorului de linii i se atribuie valoarea 1, iar valoarea contorului de pagini este mărită cu 1.

Clauza NEXT GROUP nu poate fi specificată în rubricile ce descriu grupe de tip PAGE HEADING, PAGE FOOTING şi REPORT FOOTING.

*Clauza* USAGE indică faptul că unicul mod de reprezentare a valorilor datelor ce compun linia este zecimal-despachetat.

*Rubrici de descriere a liniilor de editare şi a datelor.* Prezenţa clauzei LINE într-o rubrică de format 2 indică descrierea unei linii, iar absenţa clauzei, descrierea unei date ce aparţine unei linii.

Semnificaţia şi utilizarea clauzelor BLANK, JUSTIFIED, PICTURE, USAGE, VALUE şi LINE sînt cunoscute ; pentru celelalte clauze ele sînt prezentate în continuare.

*Clauza* SOURCE precizează date ale cărei valori vor fi atribuite datei descrise. Identificatorul desemnează o dată definită în diviziunea DATA sau unul din registrele speciale : TALLY, PAGE-COUNTER sau LINE-COUNTER.

*Clauza* COLUMN indică, prin *întreg-3* ($1 \leqslant$ *întreg-3* $\leqslant 132$), numărul coloanei din cadrul liniei începînd de la care va fi înregistrată (de la stînga la dreapta) valoarea datei.

*Clauza* GROUP indică faptul că valoarea datei va fi înregistrată numai în prima linie a grupei din care face parte ; clauza nu poate figura decît în descrierea unei date elementare ce aparţine unei linii de tip DETAIL.

*Clauza* SUM precizează faptul că data reprezintă un contor pentru obţinerea totalului unei grupe de control ; clauza poate fi citată numai în rubrica de descriere a unei date numerice elementare ce aparţine unei grupe de tip CONTROL FOOTING şi trebuie să-i fie asociată clauza PICTURE.

Valorile datelor desemnate prin *identificator-2*, *identificator-3*, ... sînt adăugate contorului :
– în momentul în care este generată o grupă de tip DETAIL (a cărei descriere citează aceste date în clauza SOURCE), în cazul în care *identificator-2*, *identificator-3*, ... reprezintă date numerice elementare definite în secţiunile FILE, WORKING sau LINKAGE ;
– înaintea schimbării grupei de control în care este definit contorul, în cazul în care *identificator-2*, *identificator-3*, ... reprezintă contoare definite în aceeaşi grupă de control sau într-o grupă de control inferioară.



Scanned with OKEN Scanner

Opțiunea UPON trebuie utilizată în cazul în care *identificator-2*, *identificator-3*, ... sînt citați în clauza SOURCE în mai multe grupe de tip DETAIL ; *nume-dată-3*, *nume-dată-4* ... reprezintă numele grupelor de tip DETAIL.

*Clauza* RESET poate fi citată în rubrica de descriere a unei date elementare, definită contor prin clauza SUM. Ea precizează variabila de control (FINAL sau *nume-dată-2*) care, la schimbarea valorii, va determina inițializarea cu zero a contorului. Variabila de control *nume-dată-2* trebuie să fie de nivel superior variabilei de control citată în clauza TYPE a grupei de tip CONTROL FOOTING în care clauzele SUM și RESET apar.

Contoarele care nu fac obiectul unei clauze RESET, sînt inițializate cu zero la schimbarea grupelor de control în care sînt definite.

5.1.12.3. *Instrucțiuni pentru generarea rapoartelor.* Operația de generare a unui raport este descrisă în limbajul COBOL prin trei instrucțiuni : INITIATE, GENERATE și TERMINATE (tabelul 5.26).

*Tabelul 5.26*

**Instrucțiuni pentru generarea rapoartelor**

INITIATE

INITIATE *nume-raport-1* [, *nume-raport-2*] ...

GENERATE

GENERATE { *nume-grupă* DETAIL / *nume-raport* }

TERMINATE

TERMINATE *nume-raport-1* [, *nume-raport-2*] ...

1) *Instrucțiunea* INITIATE inițiază operația de generare a unuia sau mai multor rapoarte, adică :

– inițializează cu valoarea zero toate contoarele asociate rapoartelor prin clauza SUM și contoarele de linii (LINE-COUNTER) ;

– inițializează cu valoarea 1 contoarele de pagini (PAGE-COUNTER) ;

– verifică ordinea grupelor raportului (care trebuie să fie următoarea : REPORT HEADING, PAGE HEADING, CONTROL HEADING, CONTROL FOOTING, PAGE FOOTING, REPORT FOOTING).

2) *Instrucțiunea* GENERATE, indiferent de opțiunea specifică (*nume-grupă* DETAIL sau *nume-report*) indică, în cazul general, realizarea automată a următoarelor operații :

– poziționarea contoarelor LINE-COUNTER și/sau PAGE-COUNTER și testarea conținutului acestora, pentru generarea grupelor de tip PAGE FOOTING și/sau PAGE FOOTING ;



Scanned with OKEN Scanner

– testarea valorilor variabilelor de control pentru generarea grupelor de tip CONTROL FOOTING și/sau CONTROL HEADING;

– obținerea totalurilor în contoarele definite prin clauza SUM și inițializarea lor la apariția unei noi grupe de control;

– generarea și editarea (la prima execuție a instrucțiunii GENERATE) grupelor de tip REPORT HEADING, PAGE HEADING și CONTROL HEADING (în ordinea indicată de ierarhia variabilelor de control);

– generarea și editarea (la următoarele execuții ale instrucțiunii GENERATE), în ordinea indicată de ierarhia variabilelor de control, a grupelor de tip CONTROL FOOTING și CONTROL HEADING.

Generarea și editarea unei grupe de tip DETAIL are loc numai dacă în instrucțiunea GENERATE este citat numele unei astfel de grupe.

*Instrucțiunea* TERMINATE încheie operația de generare a unuia sau mai multor rapoarte, adică generează și editează ultimele grupe de tip CONTROL FOOTING ce corespund variabilelor de control, grupele de tip PAGE HEADING și/sau PAGE FOOTING și grupa de tip REPORT FOOTING.

Programul 5.1 reprezintă descrierea în limbajul COBOL a operației de editare a raportului prezentat în fig. 5.1.

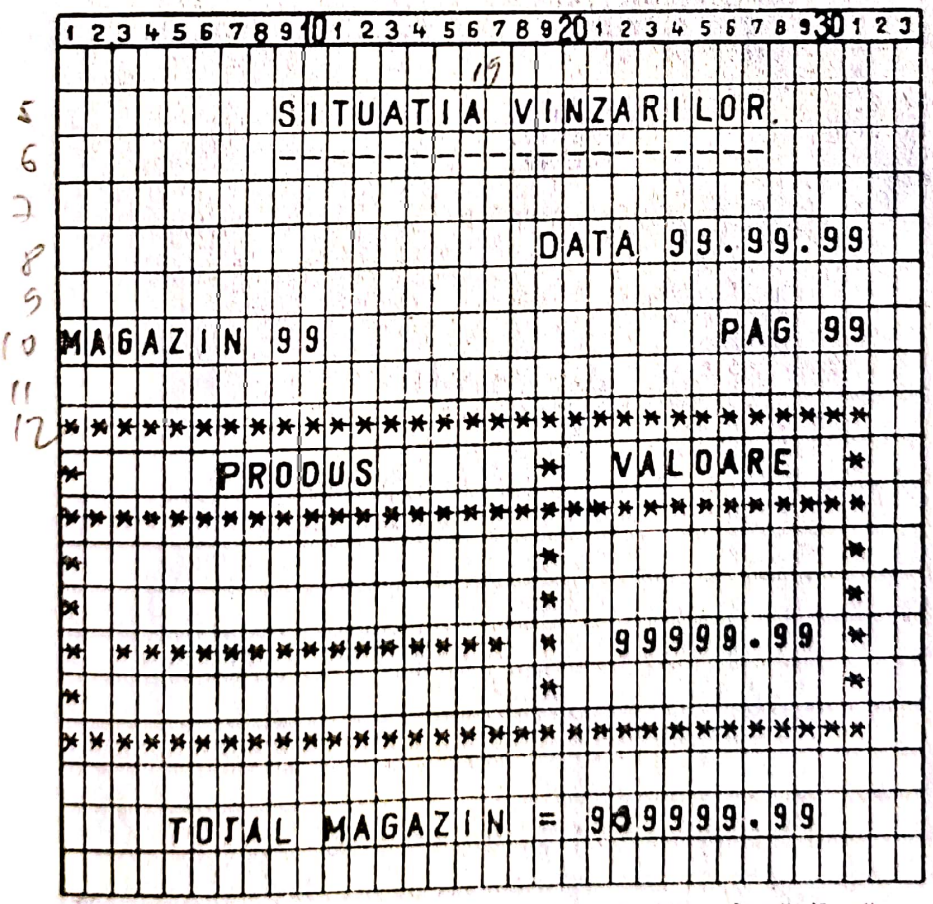



Fig. 5.1. Structura raportului „Situația vînzărilor".

## Programul 5.1

```
      JOB P51,ANIISOS,PN;FELIX
     COMPILE COBOL
*
*     PROGRAMUL REALIZEAZA PRELUCRAREA ARTICOLELOR FISIERULUI
*     FCART SI EDITAREA RAPORTULUI PREZENTAT IN FIGURA 5.1.
*     LINIA DE TIP DETAIL -L8- ESTE TIPARITA DUPA CITIREA UNUI
*     ARTICOL AL FISIERULUI FCART,IAR LINIA DE TIP CONTROL
*     FOOTING - L10,LA SCHIMBAREA GRUPEI DE CONTROL DEFINITA
*     DE DATA COD-MAGAZIN.
*
 ID DIVISION.
 PROGRAM-ID. PROG52.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FCART ASSIGN TO SYSIN.
     SELECT FIMP  ASSIGN TO SYSOUT.
 DATA DIVISION.
 FILE SECTION.
 FD  FCART RECORDING F
           LABEL RECORD OMITTED.
 01  ART.
   02 COD-MAGAZIN        PIC 99.
   02 COD-PRODUS         PIC 999.
   02 DEN-PRODUS         PIC X(15).
   02 VALOARE            PIC 9(5)V99.
 FD  FIMP  RECORDING V
           LABEL RECORD OMITTED
           REPORT IS TABEL.
 WORKING-STORAGE SECTION.
 01  DATA-ZI.
   02 ZI                 PIC 99.
   02 LUNA               PIC 99.
   02 AN                 PIC 99.
 REPORT SECTION.
 RD  TABEL CONTROL IS COD-MAGAZIN
           PAGE LIMIT IS 15
           HEADING        5
           FIRST DETAIL   10
           LAST DETAIL    12
           FOOTING        15.
 01  G1   TYPE IS REPORT HEADING NEXT GROUP NEXT PAGE.
   02 L1   LINE NUMBER IS  5.
           COLUMN IS      10
           PIC IS         X(19)
           VALUE          'SITUATIA VINZARILOR'.
   02 L2   LINE NUMBER IS  6
           COLUMN IS      10
           PIC IS         X(19)
           VALUE IS       ALL '-'
   02 L3   LINE NUMBER IS  7.
     03    COLUMN IS      20
           PIC IS         X(5)
           VALUE IS       'DATA'.
     03    COLUMN IS      25
           PIC IS         99
           SOURCE IS      ZI.
     03    COLUMN IS      27
           PIC IS         X
           VALUE IS       '.'.
     03    COLUMN IS      28
           PIC IS         99
           SOURCE IS      LUNA.
     03    COLUMN IS      30
           PIC IS         X
           VALUE IS       '.'.
     03    COLUMN IS      31
```

```
                PIC IS          99
                SOURCE IS       AN.
      01  G2P TYPE IS PAGE HEADING.
        02 L4P LINE NUMBER IS 5.
          03   COLUMN IS       2
               PIC IS          X(8)
               VALUE IS        'MAGAZIN '.
          03   COLUMN IS       10
               PIC IS          99
               SOURCE IS       COD-MAGAZIN.
          03   COLUMN IS       27
               PIC IS          X(4)
               VALUE IS        'PAG '.
          03   COLUMN IS       31
               PIC IS          99
               SOURCE IS       PAGE-COUNTER.
        02 L5P LINE NUMBER IS 6
               COLUMN IS       2
               PIC IS          X(31)
               VALUE IS        ALL '*'.
        02 L6P LINE NUMBER IS 7
               COLUMN IS       2
               PIC IS          X(31)
               VALUE IS        '*      PRODUS      *  VALOARE  *'.
        02 L7P LINE NUMBER IS 8
               COLUMN IS       2
               PIC IS          X(31)
               SOURCE IS       L5P.
      01  G3  TYPE IS DETAIL.
        02 L8  LINE NUMBER IS PLUS 1.
          03   COLUMN IS       2
               PIC IS          XX
               VALUE IS        '* '.
          03   COLUMN IS       4
               PIC IS          X(15)
               SOURCE IS       DEN-PRODUS.
          03   COLUMN IS       19
               PIC IS          X(4)
               VALUE IS        ' *  '.
          03   COLUMN IS       23
               PIC IS          Z(4)9,99
               SOURCE IS       VALOARE.
          03   COLUMN IS       31
               PIC IS          XX
               VALUE           ' *'.
      01  G4C TYPE IS CONTROL FOOTING COD-MAGAZIN.
        02 L9C LINE NUMBER IS PLUS 1
               COLUMN IS       2.
               PIC IS          X(31)
               SOURCE IS       L5P.
        02 L10 LINE NUMBER IS PLUS 1.
          03   COLUMN IS       6
               PIC IS          X(16)
               VALUE IS        'TOTAL MAGAZIN = '.
          03   COLUMN IS       22
               PIC IS          Z(5)9,99
               SUM             VALOARE.
PROCEDURE DIVISION.
1.
    OPEN INPUT FCART OUTPUT FIMP
    ACCEPT DATA-ZI FROM DATE
    INITIATE TABEL.
2.
    READ FCART AT END GO TO 3.
    GENERATE G3 GO TO 2.
3.
    TERMINATE TABEL
    CLOSE FCART FIMP
    STOP RUN.
```



Scanned with OKEN Scanner

```
.          LINK
.          RUN
01001AAAAAAAAAAAAAAAA1111111
01002BBBBBBBBBBBBBBBB2222222
01003CCCCCCCCCCCCCCCC3333333
02001DDDDDDDDDDDDDDD4444444
02002EEEEEEEEEEEEEEE5555555
.EOF
.          EOJ
```

Datele prelucrate (care reprezintă valorile obținute din desfacerea produselor magazinelor unei întreprinderi comerciale) aparțin fișierului pe cartele MAGAZIN, ale cărui articole sînt ordonate crescător după valorile datei COD-MAGAZIN, definită variabilă de control.

## 5.2. LIMBAJUL LPG

5.2.1. **Introducere.** Limbajul LPG (Langage de Programmation de Gestion) este destinat rezolvării aplicațiilor din gestiunea clasică, caracterizate prin calcule puține și multe operații de intrare/ieșire, care au ca finalitate editări de rapoarte. Datele de intrare pentru astfel de aplicații aparțin, de obicei, unuia sau mai multor fișiere, iar datele de ieșire reprezintă un raport sau mai multe rapoarte create simultan pe diferite suporturi.

Caracteristic unui text sursă LPG este faptul că numai operațiile de prelucrare a datelor sînt descrise explicit, operațiile de intrare/ieșire fiind programate de compilator implicit, pe baza informațiilor furnizate de utilizator prin descrierea fișierelor de intrare și de ieșire.

Comparativ cu alte limbaje de programare, limbajul LPG are o sintaxă rigidă, scrierea diferitelor construcții ale limbajului fiind pozițională. Acest lucru permite o programare mai simplă decît în alte limbaje, însă limitează posibilitățile de extensie a limbajului.

În cele ce urmează este descris limbajul LPG implementat la calculatoarele de tip IRIS-50 și FELIX C-256 [120, 121].

5.2.2. **Vocabularul** limbajului LPG este alcătuit din: caractere de bază, cuvinte cheie, identificatori, constante și șabloane de editare.

1) *Caracterele de bază*, în număr de 43, sînt următoarele :

– caracterele A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y și Z, numite caractere *alfabetice* ;

– caracterele 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, numite caractere *numerice* ;

– caracterele „spațiu", &, *, –, , , .,' , numite caractere *speciale*.

Un caracter *alfanumeric* este orice caracter alfabetic sau numeric.

2) *Cuvintele cheie*, formate din maximum opt caractere alfabetice, indică operații de prelucrare, caracteristici ale fișierelor, datelor etc. ; cuvintele cheie nu sînt rezervate.



Scanned with OKEN Scanner

3) *Identificatorii* sînt utilizaţi pentru referirea fişierelor, datelor, variabilelor logice (numite indicatori), datelor definite cheie de acces la articolele fişierelor şi la elementele tablourilor. Identificatorii asociaţi fişierelor sînt formaţi din maximum opt caractere alfanumerice, iar cei asociaţi datelor, tablourilor şi subprogramelor, din maximum şase caractere alfanumerice (primul caracter fiind alfabetic). Identificatorii asociaţi variabilelor logice şi cheilor de acces sînt formaţi din două caractere numerice sau alfanumerice.

4) *Constantele* sînt de două tipuri : numerice şi nenumerice. O constantă *numerică* reprezintă un număr întreg sau real format din maximum zece cifre. O constantă *nenumerică* reprezintă o succesiune de caractere delimitată prin apostrofuri (caracterul apostrof este reprezentat, în cadrul succesiunii, prin două caractere „apostrof" succesive) ; lungimea maximă este de 24 caractere.

5) *Şabloanele* reprezintă succesiuni formate din caracterele : spaţiu, 0, *, &, ., ,, L (simbol monetar) şi simbolul CR, delimitate prin apostrofuri ; ele definesc formatul de editare a datelor numerice. Numărul maxim de caractere ce compun un şablon este 24.

### 5.2.3. Structura datelor

5.2.3.1. *Date elementare.* În limbajul LPG există două tipuri de date : *alfanumerice* şi *numerice* ; acestea pot aparţine articolelor sau pot fi date izolate.

Valorile datelor alfanumerice reprezintă şiruri de maximum 256 caractere, iar valorile datelor numerice, numere întregi sau reale formate din maximum 15 cifre.

Datele alfanumerice sînt reprezentate în cod EBCDIC, iar datele numerice, în format zecimal-împachetat.

Asupra datelor numerice pot fi definite următoarele operaţii : aritmetice, de transfer, de comparare şi de editare.

*Operaţiile aritmetice* permise sînt : adunarea, scăderea, împărţirea şi înmulţirea. Operanzii pot reprezenta valori ale unor date numerice sau constante numerice formate din maximum zece cifre. Valoarea rezultatului este atribuită unei date numerice şi poate fi, înainte de atribuire, rotunjită. În cazul operaţiei de împărţire poate fi obţinut atît citul cît şi restul.

Valoarea rezultatului unei operaţii aritmetice este comparată (automat) cu valoarea zero. Rezultatul comparării este precizat prin intermediul a trei *indicatori de calcul*, care sînt asociaţi relaţiilor : mai mic, egal şi, respectiv, mai mare. Fiecăruia dintre aceşti indicatori i se atribuie valoarea adevărat, în cazul în care relaţia care îi corespunde este adevărată, sau



Scanned with OKEN Scanner

valoarea fals, în caz contrar. Indicatorii de calcul sînt identificați în cadrul programelor prin simbolurile ce aparțin intervalelor [01, 99] și [HO, H9].

*Operațiile de transfer.* Efectuarea unei operații de transfer implică prezența a două date numite : *dată emițătoare* (care poate fi și o constantă) și *dată receptoare*. Fie $l_e$ și $l_r$ lungimile acestor date. Transferul se poate realiza cu aliniere la dreapta sau cu aliniere la stînga.

a) În primul caz, numărul de caractere transferate este egal cu $\min(l_e, l_r)$. Dacă $l_r < l_e$, atunci caracterele ce nu pot fi înregistrate sînt neglijate ; în caz contrar, conținutul locațiilor din zona receptoare nefolosite rămîne neschimbat. De asemenea, dacă data receptoare este alfanumerică, iar data emițătoare este numerică, transferul este precedat de conversia valorii datei emițătoare în format zecimal-despachetat ; în caz contrar, transferul este precedat de conversia valorii de transferat în format zecimal-împachetat (în situațiile în care datele implicate în transfer sînt de tipuri diferite, numărul maxim de caractere transferate corespunde lungimii maxime a datei numerice, zecimale fiind neglijate).

b) În cel de-al doilea caz, regulile de transfer sînt asemănătoare celor prezentate cu următoarele observații :

– alinierea caracterelor în zona receptoare se face la stînga ;

– dacă data emițătoare este numerică, semnul datei emițătoare este neglijat în cazul în care $l_r > l_e$ sau este întregistrat în primele patru poziții binare ale ultimei locații în caz contrar.

*Operațiile de comparare* realizează testarea unei condiții. În limbajul LPG pot fi descrise trei tipuri de teste : testul de relație, testul de semn și testul de clasă.

a) *Testul de relație* stabilește relația în care se află valorile a două date (dintre care una poate fi și o constantă) de același tip : numerice sau alfanumerice. Dacă lungimile datelor nu coincid, testarea este precedată de completarea valorii datei de lungime mai mică cu :

– zerouri la stînga și/sau la dreapta mărcii zecimale în cazul în care data este numerică ;

– spații la dreapta în cazul în care data este alfanumerică.

Rezultatul comparării valorilor celor două date este precizat prin intermediul a trei *indicatori de relație*, care sînt asociați relațiilor mai mic, egal și, respectiv, mai mare. Fiecăruia dintre acești indicatori i se atribuie valoarea adevărat, în cazul în care relația care îi corespunde este adevărată, sau valoarea fals, în caz contrar.

b) *Testul de semn* stabilește dacă în prima locație a zonei rezervate unei date alfanumerice este înregistrat un caracter a cărui reprezentare internă aparține uneia din următoarele mulțimi ;

– $\{50\} \cup [CO, CF]$ ;
– $\{60\} \cup [DO, DF]$ ;
– $[OO, 4F] \cup [51, 5F] \cup [61, BF] \cup [EO, FF]$.



Scanned with OKEN Scanner

Valoarea 50 corespunde supraperforației 12 și caracterului &, iar valoarea 60, supraperforației 11 și caracterului – ; aceste două valori corespund modalităților de reprezentare a semnului în mecanografie. În intervalul [CO, CF] sînt incluse valorile ce corespund numerelor + 0, +1, ..., +9, iar în intervalul [DO, DF], valorile ce corespund numerelor –0, –1, ..., –9.

Rezultatul comparării în cazul testului de semn este precizat prin intermediul a maximum trei *indicatori de semn*, care sînt asociați, fiecare, uneia dintre cele trei mulțimi. Fiecăruia dintre acești indicatori i se atribuie valoarea adevarat, în cazul în care caracterul testat aparține mulțimii respective, sau valoarea fals, în caz contrar.

c) *Testul de clasă* este realizat în momentul transferării valorii unei date din zona-tampon în zona-articol, stabilind :

– semnul datei și/sau dacă această valoare a datei este formată numai din caractere zero, în cazul în care data este numerică ;

– dacă valoarea datei este formată numai din caractere spațiu, în cazul în care data este alfanumerică.

Rezultatul unui test de clasă este precizat prin intermediul a trei indicatori – asociați relațiilor mai mic decît zero, mai mare decît zero și, respectiv, egal cu zero – în cazul testării valorii unei date numerice, sau prin intermediul unui singur indicator asociat relației egal cu spații, în cazul testării valorii unei date alfanumerice. Fiecăruia dintre acești indicatori, numiți *indicatori de clasă*, i se atribuie valoarea adevărat, în cazul în care relația care îi corespunde este adevărată, și valoarea fals, în caz contrar.

Observație. Indicatorii de relație, de semn și de clasă sînt identificați în cadrul programelor prin simboluri ce aparțin intervalelor [01, 99] și [H1, H9].

*Operațiile de editare* pot fi aplicate numai datelor numerice și constau în pregătirea lor pentru tipărire. Există două modalități de a descrie operațiile de editare : prin coduri de editare și prin șabloane de editare.

a) *Codurile de editare* sînt Z, Y, X, 1, 2, 3, 4, A, B, C, D, J, K, L și M, iar semnificația lor este următoarea :

– Z indică înlocuirea fiecărui zero nesemnificativ al unui număr întreg, reprezentat zecimal-despachetat, cu caracterul spațiu ;

– Y indică editarea unui număr întreg, format din șase cifre, sub forma zz/11/aa, unde zz reprezintă ziua în cadrul lunii, 11 luna, iar aa ultimele două cifre ale anului ;

– X indică înlocuirea valorii semnului unui număr întreg pozitiv, reprezentat zecimal-împachetat, cu valoarea $F_{16}$ și apoi conversia acestui număr în format zecimal-despachetat ;

– 1 indică înlocuirea fiecărui zero nesemnificativ al unui număr real cu caracterul spațiu, inserarea virgulei ca marcă zecimală și inse-



Scanned with OKEN Scanner

rarea caracterului punct după fiecare grupă de trei cifre, începînd de la marca zecimală, spre stînga ;

– 2, 3 și 4 indică înlocuirea fiecărui zero nesemnificativ al unui număr întreg sau real cu caracterul spațiu și, eventual, inserarea virgulei ca marcă zecimală ; în plus, codurile 2 și 4 indică înlocuirea fiecărui zero al numărului cu caracterul spațiu, dacă valoarea numărului este egală cu zero ;

– A, B, C și D au aceeași semnificație ca și codurile 1, 2, 3 și, respectiv, 4 ; în plus, acestea indică adăugarea, la dreapta numărului, a caracterelor CR, dacă numărul este negativ, sau a două caractere spațiu, dacă numărul este pozitiv ;

– J, K, L și M au aceeași semnificație ca și codurile 1, 2, 3 și, respectiv, 4 ; în plus, aceste coduri indică adăugarea, la dreapta numărului, a caracterului –, dacă numărul este negativ, sau a caracterului spațiu, dacă numărul este pozitiv.

b) *Șabloanele de editare.* Semnificația simbolurilor care pot alcătui un șablon este următoarea :

– spațiu indică poziția unei cifre a numărului care, dacă este nesemnificativă, va fi înlocuită cu caracterul spațiu, sau cu caracterul *, dacă acesta figurează în șablon ;

– zero indică poziția pînă la care zerourile nesemnificative vor fi înlocuite cu caracterul spațiu (cu condiția ca acest simbol să nu fie precedat de simbolul *, iar numărul de caractere spațiu care îl preced să nu fie mai mare decît numărul de cifre ale numărului de editat) ;

– * indică poziția pînă la care zerourile nesemnificative ale numărului vor fi înlocuite cu caracterul * (cu condiția ca acest simbol să nu fie precedat de simbolul zero, iar numărul de caractere spațiu care îl preced să nu fie mai mare decît numărul de cifre ale numărului de editat) ;

– L (simbol monetar) poate apărea ca prim simbol al șablonului sau imediat în stînga simbolului zero sau * ; în primul caz, indică inserarea caracterului L în prima poziție a valorii de editat, iar în al doilea caz, inserarea caracterului L imediat în stînga primei cifre semnificative ;

– –/CR poate apărea (imediat sau nu) numai după ultimul simbol spațiu, zero sau * al șablonului, indicînd adăugarea după ultima cifră a numărului a caracterului –/a caracterelor CR, dacă numărul este negativ, sau a caracterului spațiu /a două caractere spațiu, dacă numărul este pozitiv ;

–, indică poziția în care va fi inserat caracterul , ;

– . indică poziția în care va fi inserat caracterul . (cu condiția ca în stînga sa să existe cifre editate) ;

– & poate apărea numai în stînga simbolurilor spațiu, zero, *, – sau CR, indicînd inserarea unui caracter spațiu.



Scanned with OKEN Scanner

Observații. 1. Numărul simbolurilor spațiu, zero și * din șablon trebuie să fie egal cu numărul de cifre ale numărului de editat.

2. Simbolurile din șablon care nu coincid cu poziția unei cifre, sînt inserate în poziția în care apar, sau sînt înlocuite prin caracterul spațiu sau *.

3. Editarea prin intermediul șabloanelor nu realizează alinierea la marca zecimală.

4. În definirea unei operații de editare este permisă utilizarea combinată a codurilor de editare 1, 2, 3, 4, A, B, C, D, J, K, L, M și a șabloanelor de editare 'F' și '*'. În acest caz, simbolul F indică inserarea simbolului monetar în prima poziție a valorii de editat, iar simbolul *, înlocuirea zerourilor nesemnificative cu caracterul spațiu.

5.2.3.2. *Fișiere.* În limbajul LPG pot fi definite trei tipuri de organizare logică a fișierelor: secvențială, secvențială-indexată și selectivă. Formatul articolelor poate fi fix sau variabil. În vederea prelucrării, care se realizează numai la nivel logic, compilatorul rezervă fiecărui fișier o zonă-articol, și o zonă tampon (în cazul fișierelor de organizare selectivă) sau două zone-tampon (în cazul celorlalte tipuri de organizare a fișierelor).

a) *Fișiere secvențiale.* Trei tipuri de operații pot fi definite relativ la fișierele secvențiale: *consultare*, *creare* și *actualizare*.

*Consultarea fișierelor secvențiale.* Această operație necesită ca articolele fișierului să fie ordonate — crescător sau descrescător — după valorile uneia sau mai multor date ale articolelor, numite *chei de control*. Numărul maxim de chei de control ce pot fi asociate unui fișier este nouă, iar suma lungimilor zonelor ce le sînt rezervate nu trebuie să depășească 256 locații.

Articolele fișierului pentru care valorile unei chei de control sînt egale, alcătuiesc o *grupă de control* (sau grupă verticală). În vederea identificării grupelor de control, fiecărei chei de control $C_i$ i se asociază un *indicator de control*, ce poate fi referit în cadrul programului prin indentificatorul $L_i$; $i$ reprezintă una din cifrele de la 1 la 9, care definește prioritatea cheii $C_i$ în raport cu celelalte. Prioritatea cea mai mică o are cheia de control căreia îi corespunde indicatorul de control $L_i$. Stabilirea grupelor de control se realizează prin compararea succesivă, după citirea fiecărui articol al fișierului, a valorilor cheilor sale cu cele ale articolului citit anterior, începînd cu cheia cu prioritatea cea mai mare. În urma comparării valorilor unei chei de control $C_i$, indicatorul $L_i$ care îi corespunde, precum și indicatorilor de control ce corespund cheilor cu prioritate mai mică decît a cheii $C_i$, li se atribuie valoarea adevărat (care semnifică schimbarea grupei de control), dacă cele două valori comparate nu sînt egale; celorlalți indicatori de control, ce corespund cheilor cu prioritate mai mare decît a cheii $C_i$, li se atribuie valoarea fals.

Valorile indicatorilor de control pot condiționa efectuarea anumitor operații (de exemplu, tipărirea de totaluri în momentul terminării unei grupe de control).

În cazul fișierelor cu mai multe tipuri de articole, identificarea acestora în cursul prelucrării necesită prezența, în cadrul articolelor, a uneia



Scanned with OKEN Scanner

sau mai multor date comune care alcătuiesc un *cod de identificare*. Codul de identificare are valori distincte pentru fiecare tip de articol; aceste valori sînt indicate în program.

În vederea identificării, fiecărui tip de articol îi este asociat un *indicator de intrare*, care poate fi referit în cadrul programului printr-un număr format din două cifre, cu valori în intervalul [01, 99]. Identificarea se realizează după citirea fiecărui articol și constă în compararea succesivă a valorii codului de identificare cu valorile asociate, prin program, tipurilor de articole. În urma comparării, indicatorului de intrare ce corespunde tipului de articol pentru care valoarea codului de identificare este egală cu una din valorile asociate tipurilor de articole, i se atribuie valoarea adevărat, iar celorlalți indicatori de intrare valoarea fals. Situația în care articolul citit nu corespunde nici unui tip de articol este considerată eroare și este consemnată prin atribuirea valorii adevărat unui *indicator de eroare*, identificat prin simbolul HO.

Ordinea de apariție a articolelor de tipuri diferite în cadrul unei grupe de control este indicată prin intermediul unor *indicatori de secvență*. Fiecărui tip de articol îi este asociat un indicator de secvență, identificat în program printr-un:

— simbol format din două caractere alfabetice, în cazul în care articolul de tipul respectiv are o poziție oarecare în cadrul grupei de control;

— număr format din două cifre cu valori în mulțimea $\{01, \ldots, 99\}$, în cazul în care articolul de tipul respectiv are o poziție bine definită în cadrul grupei de control; numerele asociate tipurilor de articole ca indicatori de secvență trebuie să constituie o listă ordonată crescător, primul număr din listă trebuind să corespundă tipului de articol care apare primul în cadrul grupei de control.

Consultarea simultană a mai multor fișiere necesită ca unul dintre acestea să fie definit *fișier principal*, iar celelalte, *fișiere secundare*; în general, fișierul principal este un fișier permanent, iar fișierele secundare sînt fișiere de mișcări (tranzacții). De asemenea, articolele fișierelor consultate simultan trebuie să fie ordonate — crescător sau descrescător — după valorile unei *chei de concordanță*. Cheia de concordanță poate fi alcătuită din una sau mai multe date ale articolelor (maximum nouă date). Fiecăreia dintre aceste date i se asociază un simbol $M_i$, $i \in \{1, 2, \ldots, 9\}$, care indică apartenența datei la cheia de concordanță și prioritatea datei, în raport cu celelalte, la formarea cheii de concordanță. Valorile cheii de concordanță se obțin prin concatenarea datelor care o compun, concatenare care se realizează în ordinea inversă a simbolurilor $M_i$. De exemplu, în cazul unei chei de concordanță formată din trei date, cărora le corespund simbolurile $M_1$, $M_2$ și $M_3$, ordinea concatenării este $M_3$, $M_2$, $M_1$.

Articolele fișierelor consultate simultan, pentru care valorile cheii de concordanță sînt egale, alcătuiesc o *grupă orizontală*. Citirea unui articol



Scanned with OKEN Scanner

ce aparţine unei astfel de grupe este consemnată prin atribuirea valorii adevărat unui *indicator de concordanţă*, identificat prin simbolul MR. În cazul în care valorile cheii de concordanţă pentru un anumit fişier nu apar în ordinea specificată în program, indicatorului de eroare HO i se atribuie valoarea adevărat.

Tehnica de consultare a fişierelor secvenţiale prin intermediul cheilor de concordanţă realizează o interclasare logică a fişierelor şi permite, prin testarea valorilor indicatorului de concordanţă MR, aplicarea unor prelucrări diferenţiate articolelor ce aparţin grupelor orizontale şi articolelor ce nu aparţin acestor grupe. De asemenea, această tehnică permite prelucrarea consecutivă a articolelor ce alcătuiesc grupele orizontale.

Citirea mărcii standard de sfîrşit de fişier, în cursul operaţiei de consultare, este consemnată prin atribuirea valorii adevărat tuturor indicatorilor de control $L_i$ asociaţi fişierului, precum şi unui indicator special LR, numit *indicator de sfîrşit de fişier*.

*Crearea fişierelor secvenţiale.* În principiu, scrierea unui articol într-un fişier are loc după prelucrarea unui articol citit anterior dintr-un alt fişier. În anumite situaţii, scrierea articolului poate fi condiţionată de valorile (adevărat sau fals) unor indicatori, numiţi *indicatori de ieşire*. Un indicator de ieşire poate fi oricare dintre indicatorii LPG (indicatori de control, de concordanţă, de semn etc.).

*Actualizarea fişierelor secvenţiale.* Două tipuri de operaţii pot fi realizate în cadrul actualizării : modificări şi ştergeri de articole. Ambele tipuri de operaţii trebuie să fie precedate de citirea articolelor care urmează a fi modificate sau şterse. Aceasta implică faptul că fişierele principale, prelucrate prin tehnica concordanţei, nu pot fi actualizate, întrucît sînt consultate înaintea fişierelor secundare. Rescrierea articolelor în fişier, care este cerută atît de operaţia de modificare cît şi de cea de ştergere, poate fi condiţionată de valorile unuia sau mai multor indicatori de ieşire.

b) *Fişiere secvenţiale-indexate.* Două tipuri de operaţii pot fi efectuate asupra fişierelor secvenţiale-indexate : consultare şi actualizare. Acestea pot fi realizate atît în acces secvenţial cît şi în acces direct.

Consultarea în acces secvenţial poate fi *totală* sau *parţială*. Consultarea parţială necesită existenţa unui *fişier* RAF (RECORD ADDRESS FILE), ale cărui articole (de format fix) conţin, fiecare, două chei : cheia primului articol şi cheia ultimului articol ce aparţine părţii de fişier ce va fi consultată. Într-un program nu poate fi definit decît un singur fişier RAF.

Consultarea în acces direct necesită furnizarea valorilor cheii articolelor. Această cheie, numită *cheie simbolică*, reprezintă o dată ce aparţine unui alt fişier definit în program. Cheia simbolică poate fi : alfanumerică



Scanned with OKEN Scanner

sau numerică. În cazul cheilor numerice, citirea este precedată de conversia valorilor acestora în format zecimal-despachetat. În cadrul unui program LPG pot fi definite maximum nouă chei simbolice, identificate prin simbolul $C_i$, $i \in \{1, 2, \ldots, 9\}$, care permit consultarea a maximum nouă fișiere secvențiale-indexate distincte.

Actualizarea fișierelor secvențiale-indexate poate fi realizată în acces secvențial (ștergeri și modificări de articole) sau în acces direct (ștergeri, modificări și adăugări de articole). Ștergerea sau modificarea articolelor trebuie să fie precedată de citirea (în acces secvențial sau în acces direct) acestora.

c) *Fișiere selective.* Operațiile ce pot fi efectuate asupra fișierelor selective sînt consultarea și actualizarea (modificări, ștergeri și adăugări de articole), care pot fi realizate numai în acces direct. Efectuarea unor astfel de operații necesită furnizarea a două informații :

– valoarea cheii articolului care va fi citit/scris ;

– numărul căsuței din/în care va fi citit/scris articolul.

Ca și în cazul fișierelor secvențiale-indexate, cheia articolului se numește cheie simbolică, reprezintă o dată alfanumerică sau numerică ce aparține articolelor unui alt fișier și poate fi identificată prin simbolul $C_i$, $i \in \{1, 2, \ldots, 9\}$.

Numărul căsuței trebuie stabilit în cadrul unui subprogram, scris în limbajul de asamblare.

5.2.3.3. *Tablouri.* În limbajul LPG un tablou este creat, la începutul execuției programului, prin consultarea unui fișier secvențial ale cărui articole conțin, fiecare, un număr fix de elemente (face excepție ultimul articol pentru care numărul elementelor poate fi mai mic sau egal cu numărul de elemente ale celorlalte articole). Dacă se notează cu $n_a$, numărul de articole ce compun fișierul, cu $n_e$, numărul maxim de elemente conținute de un articol și cu $n_u$, numărul de articole conținute de ultimul articol ($n_u \leqslant n_a$), atunci, numărul total de elemente ale tabloului este dat de relația

$$(n_a - 1) \times n_e + n_u.$$

Valorile elementelor unui tablou pot fi ordonate (crescător sau descrescător) sau nu.

Există două tipuri de tablouri : simple și duble. Un *tablou simplu* conține un număr fix de elemente, de aceeași lungime și tip. Un *tablou dublu* este constituit din două tablouri simple ale căror elemente alternează în cadrul articolelor, adică elementele de rang impar alcătuiesc primul tablou simplu, iar elementele de rang impar, al doilea tablou simplu. În cazul tablourilor duble, articolele fișierelor trebuie să conțină un număr par de elemente.



Scanned with OKEN Scanner

Trei tipuri de operaţii pot fi efectuate asupra tablourilor : consultare, actualizare şi extragere.

*Consultarea* constă în exploatarea elementelor tabloului (începînd cu primul element) în vederea selectării unui element care îndeplineşte o anumită condiţie. În cazul în care elementele tabloului nu sînt ordonate, condiţia poate testa numai relaţia egal.

*Actualizarea* constă în modificarea unuia sau mai multor elemente ale tabloului. Pentru realizarea acestei operaţii, tabloului îi este asociată o dată specială, a cărei valoare, la un anumit moment al execuţiei programului, reprezintă rangul ultimului element selectat. Valoarea acestui element poate fi modificată prin intermediul unei operaţii aritmetice sau de transfer, dacă numele tabloului este citat ca operand în instrucţiunea ce descrie operaţia respectivă.

*Extragerea* constă în crearea (imediat înaintea sfîrşitului execuţiei programului), din elementele tabloului, a unui fişier secvenţial, ale cărui articole conţin acelaşi număr de elemente ca şi articolele fişierului din care a fost creat tabloul.

5.2.3.4. *Rapoarte.* Un raport reprezintă un ansamblu de linii (articole) tipărit la imprimantă sau înregistrat pe un suport magnetic, în vederea unei tipăriri ulterioare.

Liniile raportului pot fi de trei tipuri :

— linii de tip H, care conţin elemente de identificare a raportului sau a paginilor acestuia ;

— linii de tip D, care conţin datele obţinute prin prelucrarea unui articol ce aparţine unui fişier de intrare ;

— linii de tip T, care conţin, de obicei, totaluri (de exemplu, totaluri ale grupelor de control).

În principiu, scrierea unei linii de tip D (care formează conţinutul principal al unui raport) are loc după citirea şi prelucrarea unui articol al unui fişier de intrare. În anumite situaţii, scrierea poate fi însă condiţionată de valorile unuia sau mai multor indicatori de ieşire. Liniile de tip H sînt scrise la începutul raportului sau la începutul paginilor, iar liniile de tip T, în momentul în care din fişierul de intrare este citit un articol diferit (ca cheie, tip etc.) de cel citit anterior.

În cazul în care raportul ocupă mai multe pagini, acesta poate conţine şi linii particulare, la căror conţinut reprezintă totaluri parţiale (de sfîrşit de pagină), reporturi şi elemente de identificare a noilor pagini. Aceste linii trebuie definite în program ca linii de tip T, H şi, respectiv, D, iar scrierea lor este condiţionată de valoarea adevărat a unui *indicator de sfîrşit de pagină.* Fiecărei imprimante (sau unităţi periferice care simulează o imprimantă) îi este asociat un indicator de sfîrşit de pagină, identificat prin unul din simbolurile OA, OB, OC, OD, OE, OF sau OV ;



Scanned with OKEN Scanner

FORMULAR F

| Număr pagină | Număr linie | Format | Nume fișier | I/O/U | Tip operație: P/S/C/R/T | Tip fișier: E/ | Sfîrșit fișier: A/D | Sens ordonare: F/V | Format articole: Lungime bloc | Format articole: Lungime articol | Tip prelucrare: L | Lungime cheie | S/I | Organizare | Indicator sfîrșit de pagină | Extensie (F/L) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 2 | 3 4 5 | 6 | 7 8 9 10 11 12 13 14 | 15 | 16 | 17 | 18 | 19 | 20 21 22 23 | 24 25 26 27 | 28 | 29 30 | 31 | 32 | 33 34 | 35 36 37 38 | 39 |
| | | F | | | | | | | | | | | | | | | |
| | | F | | | | | | | | | | | | | | | |

FORMULAR E – DESCRIEREA

PRIMUL TA

| Număr pagină | Număr linie | Format | | Cheie de acces (Ci) | Nume fișier, tablou (consultat) sau nume fișier ce conține cheia de acces | Nume fișier, tablou (creat) sau nume fișier consultat prin cheia de acces | Nume tablou (primul) sau nume subprogram | Număr elemente din articol | Număr elemente din fișier |
|---|---|---|---|---|---|---|---|---|---|
| 1 2 | 3 4 5 | 6 | 7 8 | 9 10 | 11 12 13 14 15 16 17 18 | 19 20 21 22 23 24 25 26 | 27 28 29 30 31 32 | 33 34 35 | 36 37 38 39 |
| | | E | | | | | | | |
| | | E | | | | | | | |

FORMULAR L – DESCRIEREA

| Număr pagină | Număr linie | Format | Nume fișier | 1: Nr. linie | 1: Nr. canal | 2: Nr. linie | 2: Nr. canal | 3: Nr. linie | 3: Nr. canal | 4: Nr. linie | 4: Nr. canal | 5: Nr. linie | 5: Nr. canal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 2 | 3 4 5 | 6 | 7 8 9 10 11 12 13 14 | 15 16 17 | 18 19 | 20 21 22 | 23 24 | 25 26 27 | 28 29 | 30 31 32 | 33 34 | 35 36 37 | 38 39 |
| | | L | | | | | | | | | | | |
| | | L | | | | | | | | | | | |



Scanned with OKEN Scanner

– DESCRIEREA FIŞIERELOR

| Tip unitate periferică | Idex fişier | Etichete (S/E/N) | Nume subprogram | | Adăugare/ştergere (A) | | Identificare program |
|---|---|---|---|---|---|---|---|
| 40 41 42 43 44 45 46 | 47 48 49 50 51 52 | 53 | 54 55 56 57 58 59 | 60 | 61 62 63 64 65 | 66 | 67 68 69 70 71 72 73 74 | 75 76 77 78 79 80 |

TABLOURILOR ŞI A CHEILOR DE ACCES

| BLOU | | | | AL DOILEA TABLOU | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Lungime element | Reprezentare | Număr zecimale | Sens ordonare | Nume | Lungime element | Reprezentare | Număr zecimale | Sens ordonare | Comentariu | Identificare program |
| 40 41 42 | 43 | 44 | 45 | 46 47 48 49 50 51 | 52 53 54 | 55 | 56 | 57 | 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 | 75 76 77 78 79 80 |

CONFIGURAŢIEI BENZII-PILOT

| 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nr. linie | Nr. canal | Nr. linie | Nr. canal | Nr. linie | Nr. canal | Nr. linie | Nr. canal | Nr. linie | Nr. canal | Nr. linie | Nr. canal | Nr. linie | Nr. canal | Identificare program |
| 40 41 42 | 43 44 | 45 46 47 | 48 49 | 50 51 52 | 53 54 | 55 56 57 | 58 59 | 60 61 62 | 63 64 | 65 66 67 | 68 69 | 70 71 72 | 73 74 | 75 76 77 78 79 80 |



Scanned with OKEN Scanner

## FORMULAR I — DESCRIEREA ARTICOLELOR

| Număr pagină | Număr linie | Format | Nume fişier | Indicator de secvenţă | Număr articole (1/N) | Opţiune ( /0) | Indicator de intrare | Identificare articol 1: Adresă relativă | 1: Not (N) | 1: C/D/Z | 1: Caracter | Identificare articol 2: Adresă relativă | 2: Not (N) | 2: C/D/Z | 2: Caracter | Identificare articol 3: Adresă relativă | 3: Not (N) | 3: C/D/Z | 3: Caracter |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 2 | 3 4 5 | 6 | 7 8 9 10 11 12 13 14 | 15 16 | 17 | 18 | 19 20 | 21 22 23 24 | 25 | 26 | 27 | 28 29 30 31 | 32 | 33 | 34 | 35 36 37 38 | 39 | 40 | 41 |
| | | I | | | | | | | | | | | | | | | | | |
| | | I | | | | | | | | | | | | | | | | | |

## FORMULAR O — DESCRIEREA ARTICOLELOR

| Număr pagină | Număr linie | Format | Nume fişier | H/D/T/ | Avans înainte | Avans (după) | Salt: Înainte | Salt: După | Indicatori de ieşire: AND 1: Not (N) | AND 1: Not (N) | AND 2: Not (N) | AND 2: Not (N) | Nume dată | Cod de editare | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 2 | 3 4 5 | 6 | 7 8 9 10 11 12 13 14 | 15 | 16 | 17 | 18 | 19 20 | 21 22 | 23 24 25 | 26 27 28 | 29 30 31 | 32 33 34 35 36 37 | 28 | 39 |
| | | O | | | | | | | | | | | | | |
| | | O | | | | | | | | | | | | | |

## FORMULAR C — DESCRIEREA

| Număr pagină | Număr linie | Format | /$L_t$/SR | Indicatori: AND 1: Not (N) | AND 1: Not (N) | AND 2: Not (N) | AND 2: Not (N) | Operand 1 | Cod operaţie | Operand 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 2 | 3 4 5 | 6 | 7 8 | 9 10 11 | 12 13 14 | 15 16 17 | | 18 19 20 21 22 23 24 25 26 27 | 28 29 30 31 32 | 33 34 35 36 37 38 39 40 41 42 |
| | | C | | | | | | | | |
| | | C | | | | | | | | |



Scanned with OKEN Scanner

ȘI A DATELOR FIȘIERELOR DE INTRARE

| 42 | 43 | Adresa relativă | | 52 | | |

acestui indicator i se atribuie valoarea adevărat, în momentul detectării sfîrşitului unei pagini. Anumite linii ale raportului, definite ca linii de tip H sau D, pot fi tipărite la începutul raportului sau la începutul raportului şi la începutul fiecărei pagini. În primul caz, scrierea liniei are loc dacă un indicator, numit *indicator de început de pagină* şi identificat prin simbolul 1P, are valoarea adevărat (indicatorului 1P i se atribuie valoarea adevărat la începutul execuţiei programului, iar valoarea fals, imediat înaintea citirii primului articol din fişierul de intrare). În cel de-al doilea caz, scrierea liniei are loc dacă indicatorul de început de pagină 1P sau indicatorul de sfîrşit de pagină asociat raportului, are valoarea adevărat.

Condiţiile de avans al hîrtiei de imprimat sînt definite de o bandă-pilot cu 12 canale, simulată prin program. Banda-pilot reprezintă, de fapt, un tablou de corespondenţă asociat raportului, în care, pentru fiecare canal, se indică poziţia (numărul liniei), în cadrul paginii, pînă la care va avansa hîrtia de imprimat, înainte sau după tipărirea unei anumite linii. Numărul liniei asociat canalului 1 indică începutul paginii, cel asociat canalului 12, sfîrşitul paginii, iar diferenţa dintre aceste două numere, numărul de linii ce vor fi imprimate pe o pagină.

### 5.2.4. **Structura şi execuţia programelor** LPG

5.2.4.1. *Structura programelor* LPG. Un program LPG poate grupa maximum cinci tipuri de informaţii (specificaţii) prin care sînt descrise : fişierele ; tablourile şi cheile de acces la fişierele secvenţiale-indexate şi selective ; configuraţia benzii-pilot asociată fişierelor la imprimantă ; articolele şi datele fişierelor de intrare ; operaţiile de prelucrare ; articolele şi datele fişierelor de ieşire. Fiecărui tip de informaţie îi este asociat un format de descriere specific. În fig. 5.2 sînt prezentate formularele utilizate pentru codificarea acestor tipuri de informaţii :

- formularul F — descrierea fişierelor ;
- formularul E — descrierea tablourilor şi cheilor de acces ;
- formularul L — descrierea configuraţiei benzii-pilot ;
- formularul I — descrierea articolelor şi datelor fişierelor de intrare ;
- formularul C — descrierea operaţiilor de prelucrare ;
- formularul O — descrierea articolelor şi datelor fişierelor de ieşire.

Liniile acestor formulare sînt identificate prin una din literele F, E, L, I, C sau O, indicate în coloana 6.

În cadrul unui program LPG liniile prin care sînt descrise informaţiile de un anumit tip trebuie să fie grupate, iar grupele de linii astfel constituite trebuie să fie dispuse în ordinea în care au fost enumerate tipurile de formulare. Sfîrşitul unui program LPG trebuie marcat printr-o linie care conţine, în primele patru coloane, caracterele .EOF.



Scanned with OKEN Scanner

5.2.4.2. *Execuția programelor* LPG. Caracteristic unui program LPG este faptul că numai operațiile de prelucrare (calcul) sînt programate explicit, în ordinea efectuării lor, operațiile de citire/scriere fiind programate automat de compilator. De asemenea, apariția în cursul execuției programului a anumitor evenimente (ca de exemplu citirea unui articol de un anumit tip, detectarea sfîrșitului unei grupe de control, detectarea sfîrșitului unui fișier etc.) este consemnată prin atribuirea de valori indicatorilor LPG. Testarea valorilor acestor indicatori este realizată automat, permițînd condiționarea atît a operațiilor de prelucrare cît și a celor de citire/scriere.

Programul obiect generat de compilator definește un *ciclu de bază*, în care se realizează, în principiu, citirea și prelucrarea unui singur articol al unui fișier de intrare. Repetarea operațiilor ciclului de bază este realizată automat atîta timp cît există articole de citit. Execuția ciclului de bază se face în trei faze : faza de pregătire, faza de detaliu și faza de total. În cadrul unei iterații a ciclului de bază sînt executate, obligatoriu, faza de pregătire și, în funcție de rezultatele unor teste efectuate în această fază, una din fazele de detaliu sau de total.

În principiu, principalele operații efectuate în cadrul celor trei faze sînt următoarele :

— *faza de pregătire* : atribuirea valorii adevărat indicatorului de început de pagină 1P și valorii fals celorlalți indicatori LPG ; crearea tablourilor ; citirea unui articol dintr-un fișier definit fișier de intrare ; identificarea tipului acestui articol ; compararea valorilor cheilor de control ; compararea valorilor cheii de concordanță ; operații de scriere etc. ;

— *faza de detaliu* : transferul articolului citit în zona-articol ; controlul secvenței articolului ; operații de prelucrare și de scriere ;

— *faza de total* : operații de prelucrare și de scriere, efectuate atunci cînd, în faza de pregătire, este citit un articol diferit de cel prelucrat anterior (de exemplu, citirea primului articol al unei noi grupe de control).

Faza de total nu este executată, la începutul prelucrării, în cazul primului articol care nu are asociate chei de control. Această fază este însă executată în cazul articolelor care nu au asociate chei de control și sînt citite după prelucrarea primului articol care are asociate chei de control.

Execuția sistematică a unor operații în faza de total se realizează prin intermediul unui indicator special, simbolizat prin LO, a cărui valoare este adevărat în tot cursul execuției programului.

5.2.5. **Descrierea fișierelor.** Caracteristicile generale ale unui fișier sînt precizate prin intermediul unei linii de format F, a cărei structură este prezentată în tabelul 5.27.



Scanned with OKEN Scanner

*Tabelul 5.27*

**Descrierea fişierelor (format F)**

| Coloana | Semnificaţia | Observaţii |
|---|---|---|
| 7—14 | Numele fişierului | Aliniat la stînga |
| 15 | Operaţia efectuată asupra fişierului, pusă în evidenţă prin unul dintre simbolurile :<br>I — consultare<br>O — creare<br>U — actualizare | |
| 16 | Tipul fişierului, indicat prin unul dintre următoarele simboluri :<br>P — fişier principal<br>S — fişier secundar<br>C — fişier selectiv sau secvenţial-indexat, consultat prin intermediul unei chei simbolice<br>R — fişier RAF, care conţine valorile cheii simbolice pentru consultarea parţială a unui fişier selectiv-indexat<br>T — fişier-tablou, ale cărui articole conţin elementele unui tablou | |
| 17 | Două simboluri pot apărea în această coloană :<br>E — care indică faptul că execuţia programului ia sfîrşit în momentul în care este detectată marca de sfîrşit de fişier<br>spaţiu — care indică faptul că sfîrşitul execuţiei programului nu este condiţionat de detectarea mărcii de sfîrşit de fişier | Simbolul E poate fi citat numai în descrierea unui fişier de tip P, S sau R. Dacă pentru toate fişierele definite în program, în coloana 17 figurează simbolul „spaţiu", execuţia programului ia sfîrşit după citirea tuturor articolelor fişierelor de tip P, S sau R |
| 18 | Sensul ordonării valorilor cheii de concordanţă, indicat prin unul dintre următoarele simboluri :<br>A — ordonare crescătoare<br>D — ordonare descrescătoare | Simbolurile A sau D pot apărea numai în descrierea fişierelor principale şi secundare |
| 19 | Formatul articolelor fişierului, indicat prin unul dintre simbolurile :<br>F — format fix<br>V — format variabil | Formatul articolelor poate să nu fie precizat în cazul fişierelor care utilizează suporturile şi unităţile standard ale sistemului |
| 20—23 | Lungimea maximă a zonei-tampon (în număr de locaţii) | Aliniată la stînga, valori în mulţimea {1, . . . , 9999} |



Scanned with OKEN Scanner

*Tabelul 5.27* (continuare)

| | | |
|---|---|---|
| 24—27 | Lungimea maximă a zonei-articol (în număr de locaţii) | Aliniată la stînga, valori în mulţimea {1, 2, . . . , 9999} |
| 28 | Două simboluri pot apărea în această coloană :<br>spaţiu — care indică consultarea totală a unui fişier secvenţial-indexat<br>L — care indică o consultare parţială a unui fişier secvenţial-indexat | |
| 29—30 | Lungimea zonei ce corespunde cheii simbolice (în număr de locaţii) | Aliniată la dreapta<br>Informaţia este specifică fişierelor secvenţiale-indexate şi selective, consultate în acces direct, precum şi fişierelor de tip R |
| 32 | Organizarea fişierului, indicată prin unul dintre următoarele simboluri :<br>„spaţiu“ — organizare secvenţială<br>I — organizare secvenţială-indexată<br>S — organizare selectivă | |
| 33—34 | Indicatorul de sfîrşit de pagină asociat fişierului (raportului), desemnat prin unul dintre simbolurile OA, OB, OC, OD, OE, OF, OG sau OV | |
| 39 | Două simboluri pot apărea în această coloană :<br>E — în cazul în care fişierul este de tip T, de tip R, sau în cazul în care articolele fişierului conţin valorile cheii simbolice pentru consultarea în acces direct a unui fişier selectiv sau secvenţial-indexat ;<br>L — în cazul în care fişierul este la imprimantă | Simbolurile E şi L semnifică faptul că fişierului îi este asociată o linie de descriere de format E sau, respectiv, de format L |
| 40—41 | Tipul unităţii periferice afectată fişierului, indicat prin unul dintre simbolurile : CR, CP, RD, MD, MT sau PR | |
| 47—48 | Identificatorul de exploatare (idexul) atribuit fişierului, care poate fi :<br>— una dintre literele alfabetului, urmată de una dintre cifrele zecimale<br>— unul dintre simbolurile *1, *2 sau *3. | Simbolurile *1, *2 şi *3 sînt asociate fişierelor care utilizează suporturile şi unităţile standard ale sistemului :<br>*1 — cititorul de cartele<br>*2 — imprimanta<br>*3 — perforatorul de cartele |



Scanned with OKEN Scanner

*Tabelul 5.27* (continuare)

| | | |
|---|---|---|
| 53 | Absența sau prezența și tipul etichetelor fișierului, indicate prin unul dintre simbolurile :<br>spațiu — fișierul nu are etichete<br>S — fișierul are etichete standard<br>F — fișierul are etichete standard și etichete utilizator<br>N — fișierul are etichete utilizator | Fișierele pe cartele și cele la imprimantă nu au etichete<br>Fișierele pe disc au etichete standard. Fișierele pe bandă pot avea sau nu etichete standard și/sau etichete utilizator |
| 54—59 | Numele subprogramului care realizează crearea sau verificarea etichetelor utilizator | Aliniat la stînga |
| 66 | Prezența simbolului A în această coloană indică, pentru fișierele selective sau secvențiale-indexate, operația de adăugare și/sau ștergere de articole | |
| 70 | Trei simboluri pot apărea în această coloană :<br>N — care indică anularea rebobinării benzii la detectarea mărcii de sfîrșit de fișier<br>U — care indică rebobinarea benzii și demontarea rolei la detectarea mărcii de sfîrșit de fișier<br>spațiu — care indică rebobinarea benzii la detectarea mărcii de sfîrșit de fișier | |

Fișierelor la imprimantă, care reprezintă de obicei rapoarte, li se asociază și cîte o linie de format L, care precizează condițiile de avans a hîrtiei. Informațiile conținute de această linie sînt :

– numele fișierului (raportului), în coloanele 7 – 14 ;

– tabloul de corespondență între canalele benzii-pilot și pozițiile în cadrul paginii pînă la care va avansa hîrtia, înainte sau după imprimarea diferitelor linii ale raportului, în coloanele 15 – 74.

Tabloul de corespondență trebuie să definească obligatoriu canalele 1 și 12, definirea celorlalte canale fiind opțională. De asemenea, fiecărui fișier secvențial-indexat sau selectiv, consultat prin intermediul unei chei simbolice, i se asociază o linie de format E, care conține următoarele informații :

– simbolul $C_i$ $(1 \leqslant i \leqslant 9)$ care identifică cheia, în coloanele 9 – 10 ;

– numele fișierului ale cărui articole conțin cheia simbolică, în coloanele 11 – 18, aliniat la stînga ;

– numele fișierului consultat prin intermediul cheii simbolice, în coloanele 19 – 26, aliniat la stînga ;



Scanned with OKEN Scanner

— numele subprogramului care determină numărul căsuței, în coloanele 27—32, aliniat la stînga.

5.2.6. **Descrierea tablourilor.** Fișierul ale cărui articole conțin elementele unui tablou este descris, în mod obișnuit, prin intermediul unei linii de format F, care conține în coloana 16 simbolul T, iar în coloana 39, simbolul E. Acestui fișier i se asociază și o linie de format E în care se indică :

— numele, în coloanele 11—18, dacă fișierul este consultat, sau în coloanele 19—26, dacă fișierul este creat ;

— numărul elementelor conținute de un articol, în coloanele 33—35, aliniat la dreapta ;

— numărul total de elemente conținute de fișier, în coloanele 36—39, aliniat la dreapta.

Numele și tipul tabloului, tipul, lungimea, formatul de reprezentare internă și, eventual, sensul ordonării elementelor tabloului sînt prezentate în aceeași linie, de tip E, în care este descris și fișierul-tablou (tabelul 5.28).

*Tabelul 5.28*

**Descrierea tablourilor**

| Semnificația | Coloanele | | |
|---|---|---|---|
| | Tablou simplu | Tablou dublu | |
| | | Primul tablou simplu | Al doilea tablou simplu |
| Numele tabloului (aliniat la stînga) | 27—32 | 27—32 | 46—51 |
| Lungimea unui element (în număr de caractere), aliniat la dreapta | 40—42 | 40—42 | 52—54 |
| Formatul de reprezentare internă a elementelor tabloului : spațiu — reprezentare în cod EBCDIC, în cazul elementelor care reprezintă șiruri de caractere, și în format zecimal-despachetat, în cazul elementelor care reprezintă numere întregi sau reale P — reprezentare în format zecimal-împachetat, în cazul elementelor ce reprezintă numere întregi sau reale | 43 | 43 | 55 |
| Numărul de zecimale, indicat — pentru elementele care reprezintă numere reale — prin una dintre cifrele de la 0 la 9 | 44 | 44 | 56 |
| Sensul ordonării elementelor, indicat prin unul dintre simbolurile : A— crescător B— descrescător Simbolul spațiu indică faptul că elementele nu sînt ordonate | 45 | 45 | 57 |

5.2.7. **Descrierea articolelor şi a datelor.** Două tipuri de linii sînt utilizate pentru descrierea articolelor şi a datelor ce compun articolele :

– linii de format I, în cazul articolelor şi datelor ce aparţin fişierelor de intrare ;

– linii de format O, în cazul articolelor şi datelor ce aparţin fişierelor de ieşire.

Descrierea unui tip de articol al unui fişier de intrare este realizată printr-o singură linie de tip I, în cazul în care valoarea codului de identificare a articolului nu este formată din mai mult de trei caractere, sau prin mai multe linii de format I, în caz contrar. Datele ce compun articolul sînt descrise prin linii de acelaşi tip (fiecare dată fiind descrisă printr-o linie), care trebuie să urmeze liniei sau liniilor ce descriu articolul.

În cazul fişierelor de ieşire, articolele şi datele sînt descrise asemănător, cu menţiunea că un tip de articol este descris prin mai multe linii de format O succesive, dacă scrierea articolului este condiţionată de valorile a mai mult de trei indicatori de ieşire. Elementele pentru descrierea articolelor şi datelor fişierelor de intrare şi a celor de ieşire sînt prezentate în tabelul 5.29 şi, respectiv, în tabelul 5.30.

*Tabelul 5.29*

**Descrierea articolelor şi a datelor fişierelor de intrare (format I)**

| Coloana | Semnificaţia | Observaţii |
|---|---|---|
| | *Descrierea articolelor* | |
| 7—14 | Numele fişierului de care aparţine articolul | Aliniat la stînga |
| 15—16 | Indicatorul de secvenţă asociat articolului | Un indicator de secvenţă este identificat printr-unul dintre simbolurile ce aparţin intervalului [01, 99] sau [AA, ZZ] |
| 17 | Două simboluri pot apărea în această coloană :<br>1 — care precizează că grupa de control trebuie să conţină un singur articol de acest tip<br>N — care precizează că grupa de control conţine mai multe articole succesive de acest tip | |
| 18 | Două simboluri pot apărea în această coloană :<br>„spaţiu“ — care precizează că grupa de control trebuie să conţină cel puţin un articol de acest tip<br>0 — care precizează că grupa de control poate să nu conţină articolul de acest tip | |
| 19—20 | Indicatorul de intrare asociat articolului | Un indicator de intrare este identificat printr-unul dintre simbolurile ce aparţin intervalului [01, 99] |



Scanned with OKEN Scanner

*Tabelul 5.29* (continuare)

| | | |
|---|---|---|
| 21—41 | Identificarea articolului. (Valoarea codului de identificare a articolului este indicată în coloanele 27, 34, şi 41, fiind formată prin concatenarea a maximum trei caractere. Adresa relativă în cadrul zonei-articol a acestor caractere este indicată în coloanele 21—24, 28—31 şi 35—38. În coloanele 26, 33 şi 40 se indică, prin unul dintre simbolurile C, D sau Z, condiţiile de comparare a fiecărui caracter ce compune valoarea codului de identificare şi anume :<br>C — compararea celor opt biţi ai caracterului<br>D — compararea ultimilor patru biţi ai caracterului<br>Z — compararea primilor patru biţi ai caracterului<br>Compararea se realizează pentru fiecare caracter în parte, relaţia testată fiind egal — dacă în coloana 25, 32 sau 39 ce corespunde caracterului comparat figurează simbolul spaţiu, sau *diferit de* — dacă în coloana 25, 32 sau 39 ce corespunde caracterului comparat figurează simbolul N) | În cazul în care valoarea codului de identificare a articolului este formată din mai mult de trei caractere sau în cazul în care există mai multe tipuri de articole cu acecaşi structură (şi se doreşte evitarea descrierii tututor acestor tipuri de articole), descrierea articolului trebuie continuată pe linia următoare indicînd : în coloanele 14—16 operatorul logic AND — în primul caz —, sau în coloanele 14—15 operatorul logic OR — în al doilea caz — iar în coloanele 21—41 caracterele de comparare ş.a.m.d. Linia de continuare în care este citat operatorul logic OR trebuie să conţină, în coloanele 19—20, un indicator de intrare |
| | *Descrierea datelor* | |
| 53—58 | Numele datei | Aliniat la stînga |
| 44—47 | Adresa relativă în cadrul zonei-articol a primei locaţii a zonei rezervate datei | Aliniere la dreapta, valori în intervalul [1, 9999] |
| 48—51 | Adresa relativă în cadrul zonei-articol a ultimei locaţii a zonei rezervate datei | Aliniere la dreapta, valori în intervalul [1, 9999] |
| 52 | Numărul de zecimale ale valorii datei, indicat prin una din cifrele de la 0 la 9 | |
| 43 | Formatul de reprezentare externă a valorilor datei, indicat prin unul dintre simbolurile :<br>spaţiu — reprezentare în cod EBCDIC<br>*P* — reprezentare zecimal-împachetată | Formatul de reprezentare zecimal-împachetat este admis numai pentru datele ce aparţin fişierelor pe suport magnetic |
| 59—60 | Prezenţa în aceste coloane a simbolului $L_i (1 \leqslant i \leqslant 9)$ defineşte data drept cheie de control | |
| 61—62 | Prezenţa în aceste coloane a simbolului $M_i (1 \leqslant i \leqslant 9)$ defineşte data drept cheie de concordanţă sau componentă a cheii de concordanţă, iar prezenţa simbolului $C_i$ $(1 \leqslant i \leqslant 9)$, defineşte data drept cheie simbolică | |



Scanned with OKEN Scanner

*Tabelul 5.29* (continuare)

| | | |
|---|---|---|
| 63—64 | Indicatorul LPG ale cărui valori condiționează citirea în acces direct a unui fișier selectiv sau secvențial-indexat | Citirea este condiționată numai dacă data este definită drept cheie simbolică |
| 65—70 | În aceste coloane pot fi citați unul pînă la maximum trei indicatori de clasă, a căror prezență indică efectuarea unui test de clasă, în momentul transferării valorii datei din zona-tampon în zona-articol | Un indicator de clasă este identificat printr-unul dintre simbolurile ce aparțin intervalului [01, 99] sau [H1, H9] |

*Tabelul 5.30*

**Descrierea articolelor și a datelor fișierelor de ieșire (format O)**

| Coloana | Semnificația | Observații |
|---|---|---|
| | *Descrierea articolelor* | |
| 7—14 | Numele fișierului de care aparține articolul | Aliniat la stînga |
| 15 | Simbolurile H, D sau T care pot apărea în această coloană precizează faza ciclului de bază a programului în care va fi scris articolul :<br>H sau D — faza de detaliu<br>T — faza de total | Ordinea normală de descriere a diferitelor tipuri de articole (linii ale raportului) este : H, D și T |
| 17 | Numărul de interlinii cu care avansează hîrtia de imprimat înainte de scrierea articolului | Numărul de interlinii este precizat printr-una dintre cifrele de la 0 la 9 |
| 18 | Numărul de interlinii cu care avansează hîrtia de imprimat după scrierea articolului | |
| 19—20 | Numărul de interlinii cu care avansează hîrtia de imprimat înainte de scrierea articolului, definit de canalul benzii-pilot indicat în aceste coloane | Canalul benzii-pilot este precizat printr-una dintre valorile intervalului [0,1,... 12] |
| 21—22 | Numărul de interlinii cu care avansează hîrtia de imprimat după scrierea articolului, definit de canalul benzii-pilot indicat în aceste coloane | |
| 23—31 | În coloanele 24—25, 27—28 și 30—31 pot fi citați maximum trei indicatori de ieșire, ale căror valori logice — legate prin operatorul logic *și* — condiționează scrierea articolului. Valoarea logică a unui indicator este *adevărat* — dacă în coloana 23, 26 sau 29 care îi corespunde figurează simbolul spațiu — sau *fals* — dacă în coloana respectivă figurează simbolul N | Un indicator de ieșire poate fi oricare indicator LPG. În cazul în care scrierea articolului este condiționată de valorile logice a mai mult de trei indicatori de ieșire, descrierea articolului trebuie continuată pe linia următoare, specificînd în coloanele 14—16 ale acestei linii operatorul logic AND |



Scanned with OKEN Scanner

*Tabelul 5.30* (continuare)

| | | |
|---|---|---|
| 16—18 | Două simboluri pot apărea în aceste coloane :<br>— ADD care indică adăugarea articolului (într-un fișier selectiv sau secvențial-indexat)<br>— SUP — care indică ștergerea articolului (dintr-un fișier secvențial pe disc, selectiv sau secvențial-indexat) | Dacă articolul este adăugat unui fișier selectiv, cheia de identificare a articolului trebuie să fie prima dată a acestuia<br>Datele articolului de șters nu are sens să fie descrise prin linii de tip I |
| | *Descrierea datelor* | |
| 32—37 | În aceste coloane poate fi citat numele datei, care poate fi un identificator sau unul dintre cuvintele cheie PAGE, UDATE, UDAY, UMONTH sau UYEAR a căror semnificație este :<br>PAGE — definește o dată numerică de lungime patru caractere ; valoarea acestei date este mărită cu 1 înainte de scrierea articolului și este înregistrată în zona a cărei adresă este indicată în coloanele 40—43 ;<br>UDATE — definește o dată alfanumerică, de lungime nouă caractere, a cărei valoare reprezintă data zilei (sub forma *zz/ll/aa nnn*, *nnn* indicînd numărul zilei în cadrul anului) ;<br>UDAY — definește o dată numerică de lungime două caractere, a cărei valoare reprezintă ziua curentă (*zz*);<br>UMONTH — definește o dată numerică de lungime două caractere, a cărei valoare reprezintă luna curentă (*ll*) ;<br>UYEAR — definește o dată numerică de lungime două caractere, a cărei valoare reprezintă ultimele două cifre ale anului (*aa*). | |
| 40—43 | Adresa relativă în cadrul articolului a ultimului caracter al datei | |
| 44 | Formatul de reprezentare extrenă a valorilor datei, indicat prin unul dintre simbolurile :<br>„spațiu“ — reprezentare în cod EBCDIC<br>P — reprezentare zecimal-împachetată | |
| 39 | Simbolul B indicat în această coloană indică înregistrarea, în fiecare locație a zonei rezervate datei, a caracterului :<br>„spațiu“ — dacă data este alfanumerică<br>„zero“ — dacă data este numerică | |
| 38 | Formatul de editare a valorilor datei, precizat printr-un cod de editare | |
| 45—70 | Valoarea inițială atribuită datei, precizată printr-o constantă nenumerică, sau formatul de editare, precizat printr-un șablon de editare | |



Scanned with OKEN Scanner

Observații. Descrierea articolelor și datelor unui fișier principal trebuie să preceadă pe cea a articolelor și datelor fișierelor secundare. Articolele și datele fișierelor supuse operației de actualizare trebuie descrise atît prin linii de format I, cît și prin linii de format O.

5.2.8. **Instrucțiuni.** Patru tipuri de operații pot fi descrise prin intermediul instrucțiunilor LPG : operații aritmetice, operații de transfer, operații de comparare, operații de control și operații de poziționare a indicatorilor LPG. Operațiile descrise în cadrul unui program pot fi executate fie în faza de detaliu, fie în faza de total a ciclului de bază al programului. De asemenea, execuția unei operații poate fi condiționată de valorile unuia sau mai multor indicatori LPG.

*Tabelul 5.31*

**Descrierea operațiilor de prelucrare (format C)**

| Coloana | Semnificația |
|---|---|
| 7—8 | Trei simboluri pot apărea în aceste coloane :<br>spațiu — care precizează că operația va fi executată în faza de detaliu<br>$L_i$ $(0 \leqslant i \leqslant 9)$ — care precizează că operația va fi executată în faza de total<br>SR — care precizează că operația va fi executată în cadrul unui subprogram LPG |
| 9—17 | În coloanele 10—11, 13—14 și 16—17 pot fi citați unul pînă la trei indicatori LPG, ale căror valori logice, legate prin operatorul logic *și*, condiționează execuția operației. Valoarea logică a unui indicator este *adevărat* — dacă în coloana 9, 12 sau 15 care îi corespunde figurează simbolul spațiu — sau *fals* — dacă în coloana respectivă figurează simbolul N |
| 18—27 | Identificatorul primului operand al operației sau o constantă |
| 28—32 | Codul operației |
| 33—42 | Identificatorul celui de-al doilea operand al operației sau o constantă |
| 49—51 | Lungimea datei căreia i se atribuie ca valoare, valoarea rezultatului (în cazul operațiilor aritmetice și de transfer) |
| 52 | Numărul de zecimale ale valorii rezultatului, indicat prin una dintre cifrele de la 0 la 9 |
| 53 | Simbolul H indicat în această coloană precizează că valoarea rezultatului va fi rotunjită |
| 54—59 | În coloanele 54—59 pot figura unul pînă la trei indicatori de calcul, de semn sau de relație, care sînt poziționați după execuția operației |



Scanned with OKEN Scanner

O instrucțiune este definită în program prin intermediul unei linii de format C. Liniile în care sînt definite instrucțiunile executate în fiecare din cele două faze-detaliu și total — trebuie să fie grupate, liniile ce corespund fazei de total trebuind să urmeze celor ce corespund fazei detaliu.

În tabelul 5.31 este prezentată structura liniei de format C, iar în tabelul 5.32, instrucțiunile care descriu cele patru tipuri de operații.

*Tabelul 5.32*

**Instrucțiuni LPG**

| Operația | Instrucțiunea | Descrierea operației |
|---|---|---|
| | | **Instrucțiuni aritmetice** |
| Adunare | ADD | Valorile datelor citate în coloanele 18—27 și 33—42 sînt însumate, iar valoarea sumei este atribuită datei citată în coloanele 43—48 |
| | Z-ADD | Datei citate în coloanele 43—48 i se atribuie valoarea zero, la această valoare se însumează valoarea datei citată în coloanele 33—42, iar valoarea sumei este atribuită datei citată în coloanele 43—48 |
| Scădere | SUB | Valoarea datei citată în coloanele 33—42 este scăzută din valoarea datei citată în coloanele 18—27, iar valoarea diferenței este atribuită datei citată în coloanele 43—48 |
| | Z-SUB | Datei citată în coloanele 43—48 i se atribuie valoarea zero, din această valoare este scăzută valoarea datei citată în coloanele 33—42, iar valoarea diferenței este atribuită datei citată în coloanele 43—48 |
| Înmulțire | MULT | Valorile datelor citate în coloanele 18—27 și 33—42 sînt înmulțite, iar valoarea produsului este atribuită datei citată în coloanele 43—48 |
| Împărțire | DIV | Valoarea datei citată în coloanele 18—27 este împărțită prin valoarea datei citată în coloanele 33—42, iar valoarea cîtului este atribuită datei citată în coloanele 43—48 |
| | MVR | Valoarea restului obținut în urma unei împărțiri este atribuită datei citată în coloanele 43—48 (Această instrucțiune trebuie să urmeze imediat unei instrucțiuni de împărțire, pentru care nu s-a indicat rotunjirea valorii cîtului) |
| | | **Instrucțiuni de transfer** |
| (Data emițătoare este citată în coloanele 33—42, iar data receptoare, în coloanele 43—48) | | |
| Transferul valorii unei date | MOVE | Transfer cu aliniere la dreapta |
| | MOVEL | Transfer cu aliniere la stînga |



Scanned with OKEN Scanner

*Tabelul 5.32* (continuare)

| Operația | Instrucțiunea | Descrierea operației |
|---|---|---|
| | | **Instrucțiuni de comparare** |
| Test de relație | COMP | Valorile datelor citate în coloanele 18—27 și 33—42 sînt comparate — relațiile testate fiind *mai mic*, *egal* sau *mai mare* — iar indicatorului citat în coloanele 54—59 ce corespunde relației a cărei valoare este *adevărat* i se atribuie această valoare<br>Celorlalți indicatori, citați în coloanele 54—59, li se atribuie valoarea *fals* |
| Test de semn | TESTZ | Baitul ce conține semnul datei citată în coloanele 43—48 este comparat cu valorile mulțimilor:<br>{50} ∪ [CO, CF]<br>{60} ∪ [DO, DF]<br>[00, 4F] ∪ [51, 5F] ∪ [61, BF] ∪ [EO, FF],<br>iar indicatorului citat în coloanele 54—59, asociat mulțimii de care aparține baitul, i se atribuie valoarea *adevărat*.<br>Celorlalți indicatori, citați în coloanele 54—59, li se atribuie valoarea *fals* |
| | | **Instrucțiuni de control** |
| Salt necondiționat | GOTO | Execuția programului continuă cu execuția instrucțiunii căreia îi corespunde eticheta (identificatorul) indicată în coloanele 33—42. Unei instrucțiuni îi poate fi asociată o etichetă printr-o linie de format C, care precede linia în care este descrisă instrucțiunea de salt și care conține în coloanele 28—32 cuvîntul cheie TAG, iar în coloanele 18—27, eticheta |
| | | **Instrucțiuni de poziționare a indicatorilor LPG** |
| Poziționarea indicatorilor LPG | SETON | Indicatorilor citați în coloanele 54—59 li se atribuie valoarea adevărat |
| | SETOF | Indicatorilor citați în coloanele 54—59 li se atribuie valoarea fals |

5.2.9. **Prelucrarea tablourilor.** Operația de consultare a elementelor unui tablou și selectare a unui element a cărui valoare îndeplinește o anumită condiție se realizează prin intermediul instrucțiunii LOKUP. Instrucțiunii LOKUP îi pot fi asociați unul sau doi indicatori de relație, citați în coloanele 54—59. De exemplu, dacă în instrucțiune este citat un indicator de relație, în coloanele 54—55, este selectat elementul a cărui valoare este cea mai apropiată (prin valori inferioare) de valoarea datei citate în coloanele 18—27; dacă în instrucțiune sînt citați doi indicatori de relație, de exemplu în coloanele 56—59, este selectat elementul a cărui valoare este egală sau apropiată (prin valori inferioare)



Scanned with OKEN Scanner

de valoarea datei citate în coloanele 18–27. Data citată în coloanele 18–27 trebuie să corespundă ca tip și lungime cu elementele tabloului.

În cazul prelucrării unui tablou simplu, numele acestuia este citat în instrucțiune în coloanele 33–38; în cazul prelucrării unui tablou dublu, numele primului tablou simplu este citat în coloanele 33–38, iar numele celui de-al doilea tablou simplu, în coloanele 43–48.

La prima execuție a instrucțiunii LOKUP, valoarea elementului selectat (din tabloul citat în coloanele 33–38) este înregistrată într-o zonă de lucru (rezervată de compilator) și poate fi accesibilă prin intermediul unei instrucțiuni aritmetice sau de transfer în care figurează numele tabloului. Totodată, prin execuția instrucțiunii, se atribuie valoarea adevărat indicatorului de relație ce corespunde relației îndeplinite.

La următoarea execuție a instrucțiunii LOKUP, valoarea elementului selectat anterior (și, eventual, modificată) este atribuită elementului de același rang al tabloului citat în coloanele 33–38, în cazul prelucrării unui tablou simplu, sau elementului de același rang al tabloului citat în coloanele 43–48, în cazul prelucrării unui tablou dublu.

5.2.10. **Subprograme.** Două tipuri de subprograme pot fi utilizate în cadrul unui program LPG: subprograme scrise în limbajul ASSIRIS și subprograme scrise în limbajul LPG.

1) *Subprograme* ASSIRIS. Apelul acestor subprograme se realizează prin intermediul instrucțiunii EXIT. Punctul de intrare în subprogram, indicat în coloanele 33–42 ale liniei asociate instrucțiunii, trebuie să fie definit în programul LPG referire externă, iar în subprogram, definire externă. În limbajul LPG, un identificator este definit referire externă printr-o instrucțiune ULABL, identificatorul fiind citat în coloanele 43–48. Subprogramului îi pot fi transmiși, ca parametri, date sau indicatori LPG, care trebuie definiți în programul LPG definiri externe, iar în subprogram, referiri externe. În limbajul LPG, o definire externă este specificată printr-o instrucțiune RLABL, numele datelor fiind citate în coloanele 43–48, iar simbolurile indicatorilor, în coloanele 45–46, precedate de simbolul IN, în coloanele 43–44.

Revenirea din subprogram se poate realiza prinr-o instrucțiune ASSIRIS BRU *32.

2) *Subprograme* LPG. Un subprogram LPG reprezintă o secvență de instrucțiuni LPG, prima instrucțiune a secvenței fiind BEGSR, iar ultima, ENDSR. Instrucțiunea BEGSR indică numele punctului de intrare, în coloanele 18–27, iar instrucțiunea ENDSR, numele punctului de ieșire din subprogram, în coloanele 18–27. Punctul de ieșire (care este opțional) poate fi definit printr-o instrucțiune TAG, ce precede imediat instrucțiunea ENDSR. Identificatorii definiți puncte de intrare și ieșire trebuie să fie unici.



Scanned with OKEN Scanner

Apelul unui subprogram LPG se realizează prin intermediul instrucțiunii EXSR (numele punctului de intrare fiind indicat în coloanele 33–42), iar revenirea din subprogram, prin intermediul instrucțiunii ENDSR. Este permis ca un subprogram LPG să apeleze un alt subprogram (ASSIRIS sau LPG).

**Exemplu.** Programul 5.2 exemplifică descrierea, în limbajul LPG, a raportului prezentat în fig. 5.3. În program sînt descrise două fișiere : FCART — fișier pe cartele ; FIMP — fișier la imprimantă. Fișierul FCART conține trei tipuri de articole, identificate prin carcterul (1, 2 sau 3) din prima coloană a cartelei. Articolele de tip 1 conțin codul unei magazii și data zilei ; articolele de tip 2 conțin date referitoare la materialele aflate, la o anumită dată, în magazie (tip, cod magazie, cod material, stoc și preț unitar) ; articolele de tip 3 conțin date referitoare la mișcarea materialelor într-o perioadă de timp (cod magazie, cod material, tip mișcare —1= intrare, 2= ieșire, cantitate intrată/ieșită în/din magazie). Articolele fișierului sînt ordonate crescător după valorile datei cod material, iar în cadrul acestora, după tipul articolului.

Raportul este editat pe suportul afectat fișierului FIMP. Liniile de tip D conțin datele ce reprezintă stocul valoric actualizat (cod material și valoare), obținut prin adăugarea/scăderea cantităților intrate/ieșite la/din stocul inițial și înmulțirea valorilor rezultate cu prețul unitar, iar linia de tip T, totalul valoric pe magazie, obținut prin cumularea stocurilor valorice actualizate.

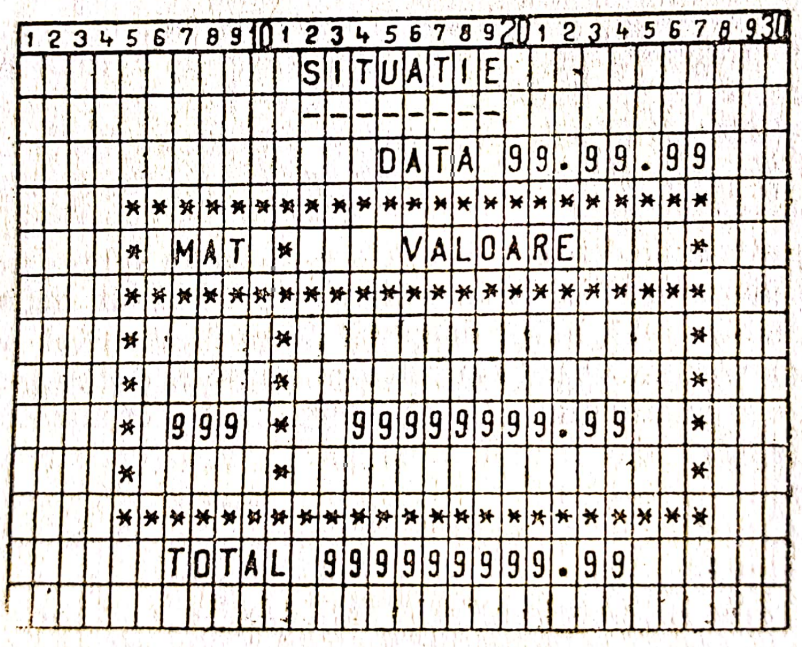



Fig. 5.3. Structura raportului „Situație".

## 5.3. LIMBAJUL RPG

Limbajul RPG (Report Programs Generator) este asemănător limbajului LPG ; fiind însă implementat pe alte sisteme de calcul (IBM, UNIVAC etc.), prezintă anumite particularități. Detalii privind acest limbaj pot fi obținute prin consultarea lucrării [109].



Scanned with OKEN Scanner

```
      JOB P052,AN:IS05,PN:FELIX
      COMPILE LPG
FFCART    I    F 256   80              CR      *1
FFIMP     O    V 278   27             LPR      *2
LFIMP       101.4812
IFCART    AA   01    1 C1
I                                                2    3 CODM
I                                                4   11 DATA
I         01N 02    1 C2
I                                                2   30COD1
I                                                4   60CODM1 L1
I                                                7  110STOC
I                                               12  162PRET
I         02N 05    1 C3
I                                                2   30COD2
I                                                4   60CODM2 L1
I                                                7   70K
I                                                8  120CANT
C    03       K          COMP 1                          12
C    03       K          COMP 2                          13
C    01                  MOVE DATA       DATE
C    01                  MOVE CODM       CODE
C    02                  MOVE PRET       PR       52
C    02                  MOVE STOC       STO      50
C    12 03    INTR       ADD  CANT       INTR     60
C    13 03    INTR       SUB  CANT       INTR     60
C    L1       STO        ADD  INTR       ST       92
C    L1       ST         MULT PR         VAL     102
C    L1       TOT        ADD  VAL        TOT     112
OFIMP.    H 9       1P
O                                       19 '--------'
O         H 2       01 1P
O                                       19 'DATA '
O                            DATE  1    27 '        '
O         H 1       01 1P
O                                       27 '*************************'
O         H 1       01 1P
O                                       27 '* MAT *     VALOARE     *'
O         H 1       01 1P
O                                       27 '*************************'
O         T 1       02 L1
O                                        6 ' * '

   O                         CODE  1     9
   O                                    13 ' *  '
   O                         VAL   1    24
   O                                    27 '  *'
   O         T 1        LR
   O                                    27 '*************************'
   O         T 1        LR
   O                                    11 'TOTAL'
   O                         TOT   1    24
.EOF
.         LINK FMS
.         RUN
10520,04,77
2050011000010020
305001111111
305001205000
2050022000020050
305002205000
305003202500
.EOF
.         EOJ
```

Programul 5.2

# 6

# LIMBAJUL PL/1

Anul 1963 a unit la lucru două grupuri de cercetători : unul, SHARE, format din utilizatori cu probleme de tip tehnico-științific, altul, GUIDE, care cuprindea utilizatori cu probleme economice ; scopul era elaborarea unui limbaj de nivel înalt care să răspundă cerințelor ambelor părți. Așa a rezultat limbajul PL/1 (Programming Language one), sinteză a ceea ce mai bun limbajele ce l-au precedat (FORTRAN, COBOL, ALGOL). Printre avantajele ce le oferă limbajul PL/1 se citează :

— ușurarea instruirii programatorilor, un singur limbaj dîndu-le posibilitatea abordării unei game variate de probleme ;
— modul de organizare a datelor ;
— instrument prețios și sintetic în asigurarea schimbului de programe ;
— creșterea eficienței utilizării resurselor sistemelor de calcul ;
— facilități în ceea ce privește dialogul om-calculator (utilizare în regim conversațional) ;
— facilități de programare modulară și structurată ;
— posibilitatea alocării dinamice a memoriei.

Pe măsură ce au trecut anii, PL/1 a căpătat o răspîndire din ce în ce mai mare. S-a dovedit foarte util, fiind folosit ca instrument de rezolvare a problemelor din cele mai variate domenii.

## 6.1. SETUL DE CARACTERE

Limbajul PL/1 admite trei tipuri de caractere : alfabetice, numerice și speciale :

< *caracter alfabetic* > : : =A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U V|W|X|Y|Z|$|

< *caracter numeric* > : : =0|1|2|3|4|5|6|7|8|9

< *caracter special* > : : = +|−|*|/|=|b|(|)|,|.|'|%|;|:|−|&|¬|>|¬<|—|?

## 6.2. STRUCTURA PROGRAMELOR

Programele PL/1 sînt constituite din două tipuri de blocuri :
– blocuri de procedură (PROCEDURE) ;
– blocuri begin (BEGIN).

Un bloc poate fi compilat separat şi linkeditat împreună cu altele deja compilate.

Orice program PL/1 este conceput ca procedură principală (MAIN PROCEDURE), execuţia, în caz că programul este constituit din mai multe blocuri, începînd cu procedura principală.

Linia de început a oricărui program este de forma :

< *etichetă* > : PROCEDURE OPTIONS (MAIN) ;

eticheta avînd cel mult şapte caractere (unele compilatoare admit cel mult şase).

Linia de sfîrşit se marchează astfel :

[< *etichetă* >] : END [< *etichetă* >]

eticheta de după END fiind cea din linia de definiţie. Linia finală poate fi multietichetată (ca în ALGOL).

## 6.3. FORMULARUL DE PROGRAMARE

PL/1 admite scrierea liberă a instrucţiunilor în coloanele 2 –72, simbolul ; marcînd sfîrşitul unei instrucţiuni. Coloana 1 rămîne liberă, iar 73 –80 sînt la dispoziţia programatorului (compilatorul nu le ia în consideraţie). Ca şi în FORTRAN, în majoritatea cazurilor, spaţiul este ignorat de compilator.

Comentariile au structura :

(/* < *şir de caractere* > */

între * şi / nefiind permis spaţiul.

## 6.4. IDENTIFICATORI, ETICHETE, CONSTANTE, VARIABILE. INSTRUCŢIUNI DE DECLARARE

Identificatorii sînt definiţi ca în COBOL şi FORTRAN (lungimea maximă 31), caracterul de început fiind alfabetic. Ca etichete se utilizează identificatorii. Există posibilitatea etichetării multiple (ca în ALGOL).



Scanned with OKEN Scanner

Identificatorii (cuvintele) cheie a căror utilizare se face ca în FORTRAN (nu ca în COBOL) sînt cuvinte cu rol special, rolul depinzînd de context. Identificatorii sînt precedați (urmați) de cîte cel puțin un spațiu sau sînt precedați (urmați) de un alt delimitator.

La baza definirii datelor în PL/1 stau atributele :

< *atribut* > : : = BINARY|DECIMAL|FIXED|FLOAT|REAL|COMPLEX|PICTURE|CHARACTER|BIT|INITIAL

Prin diverse grupări ale lor se pot obține toate tipurile de date cu care se lucrează în FORTRAN, COBOL, ALGOL. Acest lucru se face prin instrucțiuni de tip DECLARE a căror structură este

DECLARE < listă de variabile > < listă de atribute >
[,< *listă de variabile* > < *listă de atribute* >] ...

Ordinea atributelor este indiferentă, cu excepția descriptorului care nu poate sta după un nume de variabilă deoarece s-ar confunda cu dimensiunea (variabilele indexate se precizează ca în ALGOL, în cadrul listelor din DECLARE). Atributul INITIAL se utilizează pentru a inițializa valorile unor variabile. PL/1 permite dimensiuni variabile (ca în ALGOL).

Atît variabilele cît și atributele din liste sînt separate de spații. Constantele binare se deosebesc de cele zecimale prin scrierea literei B după ultima cifră a constantei.

În lipsa precizării unor atribute, variabilele care au drept prim caracter una din literele I, J, K, L, M, N sînt considerate de tip BINARY FIXED REAL (15), unde 15 reprezintă numărul de cifre binare. Variabilele care încep cu celelalte litere, implicit, sînt considerate de tip DECIMAL FLOAT REAL (6), 6 fiind numărul de cifre zecimale.

PL/1 admite informații numerice de următoarele tipuri :

– zecimal despachetat definit cu

DECLARE < *listă* > [DECIMAL] PICTURE '< *șablon* >'

șablonul dîndu-se ca în COBOL ;

– zecimal împachetat definit cu

DECLARE < *listă* > DECIMAL FIXED (*n*, *m*)

unde *n* este numărul total de cifre (inclusiv semnul) din care *m* formează partea fracționară (ca în FORTRAN) :

– zecimal virgulă mobilă definit cu

DECLARE < *listă* > DECIMAL FLOAT (*n*)

*n* fiind număr de octeți ;



Scanned with OKEN Scanner

– binar virgulă fixă definit cu

DECLARE < *listă* > BINARY FIXED (*n*, *m*)

*n* și *m* fiind numere de cifre binare ;

– binar virgulă mobilă definit cu

DECLARE < *listă* > BINARY FLOAT (*n*)

*n* fiind numărul de biți (exponentul se exprimă ca număr zecimal întreg).

Informațiile alfanumerice pot fi de tip :

– șir de caractere definite cu

DECLARE < *listă* > CHARACTER (*n*)

*n* fiind număr de caractere (octeți) ;

– șir binar definit cu

DECLARE < *listă* > BIT (*n*)

*n* fiind număr de biți.

**Exemple.** 1. Să se scrie instrucțiunea de declarare prin care un vector *V* cu zece componente are primele cinci componente egale cu 1 și următoarele cinci egale cu 0.

Dacă fiecare componentă are două cifre zecimale, convenția virgulă fixă, atunci

DECLARE V(10) DECIMAL REAL FIXED (2) INITIAL ((5)1, (5)0) ;

2. Instrucțiunea

DECLARE E(3,3) DECIMAL FIXED REAL (1,(2)0,0,1,(3)0,1) ;

definește matricea E cu 1 pe diagonala principală și cu 0 în rest.

3. Dacă se consideră

DECLARARE T(2,–1 : 1) REAL FIXED DECIMAL ((3)(0, *)) ;

înseamnă că matricea T are trei din cele șase elemente egale cu 0 și trei neprecizate.

4. Dacă se consideră declarația :

DECLARE (X INITIAL ('A'), Y INITIAL ('A' 'B'),

Z INITIAL ('ABCDEFG')) CHARACTERE (6),

K INITIAL (10100B),

V INITIAL ('0110'B) BIT (7), W INITIAL ((3)'01'B)BIT(6),

(I INITIAL (17), J INITIAL (3456)) FIXED (3),

atunci variabilele X,Y,Z,K,V,W,Q,R vor avea respectiv valorile : Abbbbb , A'Bbbb ; ABCDEF ; 000000000010100B (deoarece K este de tip implicit BINARY FIXED REAL (15)) ; 0110000 ; 010101 ; 017 ; 456 (cu b s-a notat spațiul.)



Scanned with OKEN Scanner

## 6.5. EXPRESII. INSTRUCȚIUNI DE ATRIBUIRE. STRUCTURI DE DATE

**Expresiile** sînt construite și evaluate ca în FORTRAN, operatorii utilizați fiind (în cazul utilizării alfabetului extins 60 de caractere) :

⟨ *operator numeric* ⟩ : : = + | − | / | * | * *
⟨ *operator relațional* ⟩ : : = < | <= | > | >= | ¬< | ¬> | ¬ = | =
⟨ *operator logic* ⟩ : : = ¬ | | | &
⟨ *operator de concatenare* ⟩ : : = ||

În cazul utilizării alfabetului cu 48 caractere (este cazul pentru SUBSET PL/1) operatorii sînt respectiv :

⟨ *operator relațional* ⟩ : : = LT | LE | GT | GE | NL | NG | NE | EQ
⟨ *operator logic* ⟩ : : = NOT | OR | AND
⟨ *operator de concatenare* ⟩ : : = CAT

iar în loc de ; și de : se vor scrie, respectiv, ,. și .. (două caractere.). Instrucțiunile de atribuire sînt definite ca în FORTRAN, PL/1 admițînd și atribuirea multiplă (ca în ALGOL) și permite și atribuirea alfanumerică.

Limbajul PL/1 permite structurarea datelor după modelul din COBOL, numerele de nivel fiind 1, 2, 3, ... În legătură cu structurile, este de menționat o caracteristică a lui PL/1. Dacă se consideră, de exemplu, structura

```
DECLARE  1 NUME,
         2 NUME-FAMILIE CHARACTER (10),
         2 PRENUME  CHARACTER (10);
```

instrucțiunea NUME = '*' are același efect ca instrucțiunile NUME-FAMILIE = '*' și PRENUME = '*', ceea ce conduce la completarea celor două cîmpuri cu *bbbbbbbbb. Similar se petrec lucrurile și pentru structuri de date de celelalte tipuri.

## 6.6. INSTRUCȚIUNI DE CONTROL

Instrucțiunea GO TO are structura

[⟨ *e* ⟩ :] GO TO *el*

unde *e* și *el* sînt etichete PL/1. Eticheta *el* poate fi indexată, ceea ce face ca acest tip de GO TO să fie echivalent cu GO TO calculat din FORTRAN și cu GO TO DEPENDING ON din COBOL.



Scanned with OKEN Scanner

Structura instrucţiunilor IF este

[< *e* >:] IF < *expresie condiţională* > THEN < *i1* >; [ELSE < *i2* >];

unde *i1*, *i2* pot fi instrucţiuni simple, bloc BEGIN, DO sau IF.

Limbajul PL/1 admite instrucţiuni DO de două forme. Una dintre acestea este

[< *e* >:] DO [< *v* >=< *listă* >];

.
.
.

[< *e1* >:] END [< *e* >];

unde lista conţine elemente de tipurile: listă explicită de valori sau listă implicită de forma *x*=*vi* TO *vf* [BY *r*]. Cu *x* s-a notat o variabilă, cu *vi* valoarea iniţială cu *vf* valoarea finală şi cu *r* raţia (cînd *r*=1 specificaţia BY *r* poate lipsi). Nu există restricţii de tip ca în FORTRAN.

A doua formă de DO este

[< *e* >:] DO [< *listă* >] WHILE (< *expresie condiţională* >);

. . . . . .

[< *e1* >:] END [< *e* >]

Observaţie. Este posibilă multietichetarea instrucţiunilor de control.

Exemplu

```
DECLARE ET(0:2) LABEL;
  ...
atribuire de valoare (0 sau 1) lui K;
  ...
GO TO ET (K);
ET(0): DO V=3 TO 15,0.5,-0.5,10 TO 1 BY -2;
END: V=1; GO TO ET (2);
  ...
ET (1): IF X=Y THEN IF Z=T THEN U=W THEN
V=0; ELSE V=1;
ET (2): ...
```

## 6.7. INSTRUCŢIUNI DE INTRARE/IEŞIRE

PL/1 admite două tipuri de intrări/ieşiri: în flux continuu (pentru fişierele sistem), pe înregistrări (pentru fişierele utilizator).

**6.7.1. Fişiere standard.** Pentru fişierele standard există instrucţiunile:

[< *e* >:] GET { LIST (< *listă de variabile* >) / DATA [(< *listă de variabile* >)] } [COPY];

$$[<e>:] \text{ PUT } \left[\left\{\begin{array}{l}\text{PAGE [LINE } (n)]\\ \text{SKIP [}(n)]\\ \text{LINE } (n)\end{array}\right\}\right]\left\{\begin{array}{l}\text{LIST } <\textit{listă de variabile}>\\ \text{DATA [(}<\textit{listă de variabile}>)]\end{array}\right\}$$

care permit, respectiv, citiri de liste de valori și afișări de valori.

**Exemple 1.** Fie secvența de instrucțiuni :

```
DO I=-1,0 ;
GET LIST (X,Y,Z) ; R=X+Y+Z ; PUT LIST (R) ;
END
```

Dacă pe suport se scrie 1,2,3,b,4,5, vor rezulta două valori : 1+2+3, 1+4+5. Prezența spațiului a făcut ca în al doilea triplet (ciclul are doi pași) X să rămînă cu vechea valoare (1).

Se consideră secvența de program :

```
DO I=1 TO 3 ;
    GET DATA ;
    PUT LIST (X+Y+Z) ;
END
```

Dacă pe suport se va scrie

X=10, Y=1, Z=0 ; X=4, Z=2 ; Y=3,

atunci se vor calcula trei valori : 10+1+0, 4+1+2 și 4+3+2 (la a doua valoare s-a păstrat Y anterior, iar la a treia s-au păstrat X și Z).

Observații. Opțiunile PAGE, SKIP, LINE permit, respectiv, trecerea la o nouă pagină, salt peste $n-1$ linii, trecerea la linia a $n$-a din pagina curentă.

În lista de variabile a unui GET DATA (PUT DATA) nu pot apărea componente de variabile indexate. Pentru a face ca elementele X(1,1) și X (2,2) ale unei matrice X să capete, respectiv, valorile 10, 20, se va utiliza instrucțiunea

```
GET DATA (X)
```

iar pe suport se va scrie următorul șir de caractere :

X (1,1)=10, X (2,2)=20.

Restul componentelor lui X rămîn cu vechile valori.

Pe lîngă formele de GET și PUT date, mai există :

$$[<e>:] \text{ GET EDIT } (<\textit{listă}>)\ (<\textit{format}>)\ [(<\textit{listă}>)\ (<\textit{format}>)] \ldots [\text{COPY}] ;$$

$$[<e>:] \text{ PUT } \left[\left\{\begin{array}{l}\text{PAGE LINE } (n)\\ \text{SKIP [}(n)]\\ \text{LINE } (n)\end{array}\right\}\right] \text{ EDIT } (<\textit{listă}>)\ (<\textit{format}>)\ [(<\textit{listă}>)\ (<\textit{format}>)] \ldots$$

care permit citirea/scrierea cu format (ca în FORTRAN).



Scanned with OKEN Scanner

Formatul se poate da ca listă de descriptori sau R (< *e* >), unde *e* este eticheta la care se scrie formatul (ca în FORTRAN).

Descriptorii ce se pot utiliza sînt : A, X, B (pentru şiruri de caractere), F, E (pentru zecimal virgulă fixă şi respectiv virgulă mobilă). Formele generale de utilizare ale acestora sînt : *k*A(*n*) pentru a lua de *k* ori *n* caractere alfanumerice ; X(*n*) pentru a neglija *n* caractere ; *k*B(*n*) pentru a lua de *k* ori *n* cifre binare ; *k*F (*p* [,*q*]) echivalează, în FORTRAN, cu *k*I*p* dacă lipseşte *q* şi cu *k*F*p*.*q* cînd există *q* ; *k*E (*p*, *q* [, *m*]) unde *m* este numărul de cifre ale mantisei.

**Exemple**

```
GET EDIT (K, X, L, Y) (F (2), A (K), F (3), F (L)) ;
GET LISTA (A) ;
IF A=0 THEN Y=ET1 ; ELSE Y=ET2 ;
GET EDIT (U, V, W) (R (Y)) ;
ET1 : FORMAT (2F (7,2), F (5,0)) ;
ET2 : FORMAT (3F (4,1)) ;
PUT EDIT ((X (I, J) DO J=1 TO 10) DO I=1 TO 20) (F (6,2), SKIP);
```

Observaţie. PL/1 admite factor de multiplicitate variabil, ceea ce în FORTRAN nu există. De exemplu, pentru instrucţiunea GET EDIT (X,Y,Z)((K)F(5,1), (1−K)F(6,2),(2−K)F(7,3)

În funcţie de K, descriptorii asociaţi pentru X, Y, Z sînt :

| K | X | Y | Z |
|---|---|---|---|
| K⩾2 | F (5,1) | F (5,1) | F (5,1) |
| K=1 | F (5,1) | F (7,3) | F (5,1) |
| K=0 | F (6,2) | F (7,3) | F (7,3) |
| K=−1 | F (6,2) | F (6,2) | F (7,3) |
| K⩽−2 | F (6,2) | F (6,2) | F (6,2) |

fiind necesară uneori reluarea (ca în FORTRAN).

Operaţiile de intrare/ieşire la consolă se fac cu instrucţiunea :

[< *e* >:] DISPLAY (< *mesaj* >) [REPLY (< *listă* >)]

Fără REPLY efectul este numai afişarea la imprimantă, iar cu REPLY operatorul dă datele prin tastare la maşina de scris a pupitrului de comandă.

6.7.2. **Fişiere utilizator.** Atributul RECORD este acela care pune în evidenţă acest mod de lucru. Instrucţiunile de intrare vor fi READ FILE, iar cele de scriere vor fi WRITE FILE şi REWRITE FILE.

Deoarece există multe asemănări cu lucrul în COBOL şi ALGOL, vom prezenta cîteva exemple, valabile pentru calculatoare IBM care dispun de compilator PL/1.

a) Fişier de intrare pe cartele :

```
DECLARE CARTELA
FILE RECORD INTRARE
ENVIRONMENT (MEDIUM (SYSIPT,2540)F(80)) ;
```



Scanned with OKEN Scanner

b) Fişier de intrare pe bandă :

```
DECLARE B1 FILE RECORD INPUT
ENVIRONMENT (MEDIUM(SYS010,2400)F(100)) ;
```

c) Fişier de ieşire pe bandă :

```
DECLARE B2 FILE RECORD OUTPUT
ENVIRONMENT (MEDIUM(SYS010,2400)F(300,60))
```

Aici înregistrările sînt grupate cîte cinci ($5 \cdot 60 = 300$).

Structura instrucţiunii OPEN este următoarea :

OPEN < *grup opţiuni* > [, < *grup opţiuni* >] ... ;

unde :

< *grup opţiuni* > : : = FILE < *nume fişier* > [{INPUT / OUTPUT / UPDATE}] [{STREAM / RECORD}]

[{SEQUENTIAL / DIRECT}] [{BUFFERED / UNBUFFERED}]

Instrucţiunea CLOSE are forma :

CLOSE FILE (< *nume fişier* >) [, FILE (< *nume fişier* >)] ... ;

**Exemplu de program**

```
PROG : PROCEDURE OPTIONS (MAIN) ;
       /* REZOLVAREA ECUAŢIEI DE GRADUL 2, FIIND
          DATE MAI MULTE TRIPLETE A,B,C,
          CONCEPUTE CA FIŞIER PE CARTELE */
       DECLARE (A,B,C,DELTA,U,V,X1,X2,W) REAL
               DECIMAL FLOAT (5) ; ON ENDFILE
               (SYSIN) GO TO SFIRSIT ;

       PUT LIST ('REZOLVAREA ECUAŢIEI DE GRADUL 2') ;

CIT :  GET LIST (A,B,C) ;

            IF A=0 THEN IF B=0 THEN
                  PUT LIST ('ECUATIE DEGENERATA') ;
                  ELSE BEGIN ; U=-C/B ;
                        PUT LIST ('RADACINA=',U) ;

                         END ;

                  ELSE DO;DELTA=B*B-4*A*C ;
                        V=-B/(2*A) ;
                        IF DELTA=0 THEN
                        DO;X1,X2=V ;
                           PUT LIST (X1, X2) ;
                        END
```



Scanned with OKEN Scanner

```
                    ELSE IF DELTA >0 THEN
                    DO;  W=SQORT (DELTA)/(2*A) ;
                              X1=V+W; X2=V-W ;
                              PUT LIST (X1,X2) ;
                    END ;
                    ELSE DO ; W=SQRT (-DELTA)/(2*A) ;
                              PUT LIST (V,W) ;
                              PUT LIST (V,-W) ;
                    END ;
               END ;
          GO TO CIT ;
SFIRSIT : END PROG ;
```

Scanned with OKEN Scanner

# 7

# LIMBAJE DE MACROGENERARE

Un macrogenerator (sau macroprocesor) este, în general, un translator care permite de a genera un text într-un anumit limbaj în funcție de un *model*, un *prototip* sau o *macroinstrucțiune* și de un anumit număr de parametri. Definirea modelului se face utilizînd instrucțiunile limbajului de macrogenerare (numite uneori și metainstrucțiuni). În principiu instrucțiunile limbajului de macrogenerare permit :

a) definirea modelului, a parametrilor și eventual a modului de apelare a modelului ;

b) declararea variabilelor de macrogenerare (macrovariabile), adică a variabilelor folosite în cursul procesului de generare ;

c) descrierea procesului de generare.

În practică se disting două categorii de macrogeneratoare : 1) macrogeneratoare integrate într-un compilator sau asamblor și 2) macrogeneratoare independente de orice alt translator.

În continuare se vor prezenta pe scurt limbajele de macrogenerare MAGIRIS [5, 116, 125] și LPS [126, 127] pentru care există macroprocesoare pe sistemul FELIX C–256.

7.1. **Limbajul** MAGIRIS Macrogeneratorul MAGIRIS prelucrează un text sursă (un prototip plus valorile actuale ale parametrilor) scris în limbajul MAGIRIS și generează un text obiect în unul din limbajele ASSIRIS FORTRAN, COBOL etc. Un astfel de text sursă se compune din două tipuri de instrucțiuni :

– instrucțiuni de generare ce definesc modul de generare a textului obiect și valorile actuale ale parametrilor ;

– instrucțiuni din limbajul obiect care conțin elemente ce sînt declarate ca parametri sau variabile substituabile ; aceste instrucțiuni sînt utilizate de către macrogenerator pentru a genera textul obiect.

În continuare se prezintă un exemplu de prototip (fig. 7.1) care permite generarea unei instrucțiuni, în limbaj ASSIRIS, pentru descrierea unui fișier secvențial pe disc magnetic de tip DIAM sau DIMAS. Drept parametru se folosește variabila VERSIUNE care poate avea una din valorile : 'DIAM', 'MD–25', 'MD–50', 'MD–100' funcție de

```
          MPAR      MCD=12,MARG=22
          MSTRING   VERS='DIAM','MD-25','MD-50','MD-100'
           MSTRING   DVT='RD','MD','AD',,'DC'
          MSTRING   BFS='712','1944'
          MINDEX    I=1
          MINDEX    J=4
           MARK      1000
90        MNOP
          MGO       100,&VERS(I),EQ,&VERSIUNE
          MINDEX    I=I+1
          MGO       90,I,LE,J
          MNOP
*    VERSIUNE DIFERITA DE DIAM,MD-25,MD-50,MD-100
          MGO       1000
100       MNOP
          MGO       120,I,NE,1
          MINDEX    K=1
110       MNOP
FDISC     SFD       A1,DVT:&DVT(I),PRM:INP,ACM:SEQ,BFC:AUT,                *
                    BFS:&BFS(K),BFN:2,BFA:ZTD,RCS:175,MCD:(GT),             *
                    MFF:(DLC),EOF:SF
          RES,8     0
          RES       &BFS(K)
          RES       &BFS(K)
          MGO       1000
120       MINDEX    K=2
          MGO       110
1000      MNOP
          MEND
```

Fig. 7.1. Program MAGIRIS pentru generarea descrierii unui fişier secvenţial în limbajul ASSIRIS.

tipul discului magnetic utilizat. De exemplu, dacă parametrul primeşte valoarea 'DIAM' atunci se generează secvenţa

```
FDISC     SFD       A1,DVT:RD,PRM:INP,ACM:SEQ,BFC:AUT,BFS:712,
                    BFN:2,BFA:ZTD,RCS:175,MCD:(GT),MFF:(DLC),EOF:
                    SF
ZDT       RES,8     0
          RES       712
          RES       712
```

MARK, MGO, MNOP, MINDEX sînt exemple de instrucţiuni de generare în timp ce & VERS (I), & DVT (I), & BFS (K) sînt exemple de variabile substituabile. Se precizează că o *variabilă substituabilă* este delimitată în stînga de caracterul &.

Orice zonă dintr-o instrucţiune, unde pot exista variabile substituabile este numită *parte substituabilă*. După cum se remarcă din exemplu, sintaxa instrueţiunilor MAGIRIS este similară cu cea a instrucţiunilor ASSIRIS.

**7.1.1. Elemente de bază ale limbajului** MAGIRIS. Limbajul MAGIRIS are la bază un vocabular compus din : caractere de bază, simboluri, constante, elemente, expresii aritmetice, expresii relaţionale.



Scanned with OKEN Scanner

1) *Caractere de bază* sînt : litere (A – Z) ; cifre (0 – 9) ; caractere speciale (spaţiu, virgula, +, –, *, (,) , /, :, %, ', ., &, <, ¬, #, ;, ", |, ?, !, $, –, >).

2) *Simbolurile* pot fi : etichete, mnemonice, nume de indici, nume de şiruri, nume de funcţii intrinseci.

*Etichetele* sînt succesiuni de maximum opt cifre.

*Nume de indice* este orice succesiune de maximum opt litere sau cifre, primul caracter fiind obligatoriu una din literele : I, J, K, L, M, N (iniţiale de indici). Deci :

< *nume-indice* > : : = < *iniţială-indice* > |
< *nume-indice* > < *literă* > |
< *nume-indice* > < *cifră* >

< *iniţială-indice* > : : = I | J | K | L | M | N

Un nume de indice desemnează un număr întreg cu maximum zece cifre, reprezentabil în cod complementar sub formă de cuvînt. Un nume de indice se defineşte folosind instrucţiunea MINDEX ; dacă se întîlneşte în text un nume de indice nedefinit, atunci i se asociază valoarea zero.

*Nume de şir* este o succesiune de maximum opt caractere :

< *nume-şir* > : : = < *iniţială-şir* > |
< *nume-şir* > < *literă* > |
< *nume-şir* > < *cifră* >

< *iniţială-şir* > : : = A|B|C|D|E|F|G|H|O|P|Q|R|S|T|U|V|W|X|Y|Z

Numele de şiruri desemnează succesiuni de maximum 1024 caractere. Definirea unui nume de şir se face utilizînd instrucţiunea MSTRING. Apariţia în text a unui nume de şir nedefinit desemnează şirul vid (şirul de lungime nulă).

*Numele de funcţii intrinseci* sînt în număr de patru : MC, ML, MN, MS şi sînt exclusiv definite asupra şirurilor de caractere.

3) *Constantele* pot fi numerice sau de tip şir. Constantele numerice se reprezintă ca succesiuni de maximum zece cifre, iar cele de tip şir ca succesiuni de caractere între apostrofuri ; în şir caracterul apostrof se dublează.

4) *Elementele* sînt noţiuni asociate cu noţiunea de şir. În general şirurile se definesc în MAGIRIS ca liste de elemente, fiecare listă putîndu-se reduce şi la un singur element. Deci un element este o componentă a unui şir (numit şir principal), cu următoarele restricţii :

– lungimea maximă a unui şir definit ca o listă de elemente este de 1 024 caractere ;

– numărul maxim de elemente dintr-o listă este 64.



Scanned with OKEN Scanner

5) *Funcţiile intrinseci* sînt aplicabile numai şirurilor de caractere. Există patru astfel de funcţii : MC, ML, MN, MS.

*Funcţia* MC dă numărul de caractere (sau lungimea) dintr-un şir.

*Funcţia* ML dă numărul de elemente dintr-o listă (vezi noţiunea de element).

*Funcţia* MN poate avea una din valorile : 0, 1 sau 2. Aceste valori se atribuie, ţinînd seama de şirul asupra căruia se aplică, astfel : 0, dacă primul caracter din şir este o literă sau un caracter special neurmat de o cifră ; 1, dacă primul caracter din şir este o cifră ; 2, dacă primul caracter din şir este un caracter special şi este urmat de o cifră.

Dacă funcţia MN se aplică unui şir format dintr-un singur caracter, atunci valorile se atribuie astfel : 0, dacă caracterul este o literă ; 1, dacă este o cifră ; 2, dacă este un caracter special.

*Funcţia* MS se foloseşte în cazul în care se doreşte obţinerea unui şir printr-o dublă substituire (vezi noţiunea de substituire).

6) *Expresiile aritmetice* sînt constituite din unul sau doi operanzi uniţi printr-un operator aritmetic :

$$\begin{aligned} &< \textit{expresie} > ::= < \textit{operand} > | \\ &\qquad < \textit{semn} > < \textit{operand} > | \\ &\qquad < \textit{operand} > < \textit{operator} > < \textit{operand} > \end{aligned}$$

$$< \textit{operand} > ::= < \textit{constantă numerică} > | < \textit{nume-indice} >$$

$$< \textit{semn} > ::= + | -$$

$$< \textit{operator} > ::= + | - | * | /$$

O expresie aritmetică este evaluată în cod complementar (valorile sînt reprezentate sub formă de cuvinte).

7) *Expresiile relaţionale* se definesc astfel :

$$\begin{aligned} &< \textit{expresie-relaţională} > ::= < \textit{termen} >, < \textit{operator-relaţional} >, \\ &\qquad\qquad < \textit{termen} > \end{aligned}$$

$$< \textit{termen} > ::= < \textit{expresie} > | < \textit{element} >$$

$$< \textit{operator-relaţional} > ::= \text{EQ}|\text{NE}|\text{LT}|\text{LE}|\text{GT}|\text{GE}$$

O expresie relaţională poate avea doar două valori : 1 (adevărat) şi 0 (fals) după cum relaţia este îndeplinită sau nu. Acestea servesc pentru a exprima condiţii în instrucţiunile de salt condiţionat. Termenii pot reprezenta valori întregi (cazul expresiilor) sau şiruri (cazul elementelor). Se disting astfel două tipuri de comparări :

- comparări numerice cînd ambii termeni sînt expresii ;
- comparări între şiruri cînd cel puţin un termen este un element.

În cazul în care un termen este un element, iar celălalt o expresie atunci valoarea expresiei este convertită în şir de caractere. Cele două şiruri de comparat se aduc la aceeaşi lungime prin completarea celui mai



Scanned with OKEN Scanner

scurt cu spații la dreapta, compararea făcîndu-se apoi caracter cu caracter de la stînga spre dreapta. În cazul conversiei în șir a unei valori numerice negative se introduce un semn minus la stînga primei cifre.

7.1.2. **Prelucrarea textului sursă.** Textul sursă (sau prototipul) este analizat linie cu linie, fiind tratate în mod special doar liniile care corespund instrucțiunilor de generare: MARK, MPAR, MGO, MINDEX, MSTRING, MNOP, MNOTE, MDO, MFETCH, MEND. În liniile corespunzînd unor instrucțiuni din textul obiect sînt căutate variabile substituabile, efectuîndu-se totodată și substituțiile necesare.

*O variabilă substituabilă* poate fi:

a) un nume de indice precedat de &;
b) un nume de șir precedat de &;
c) un nume de funcție intrinsecă precedat de &.

Sfîrșitul unei astfel de variabile este delimitat de unul din următoarele caractere: virgulă, spațiu, ., (,) , +, −, *, /, :, =, &, '. În cazul în care sfîrșitul simbolului este delimitat de o paranteză deschisă, variabila este considerată pînă la paranteza închisă corespunzătoare, iar dacă este caracterul punct, atunci acesta face parte din variabilă.

**Exemple de variabile substituabile:**

```
& I1                    & ALFA(J)
& ALFA                  & MC(ALFA)
& BETA(3)               & MC(BETA(4))
```

Orice secvență din textul sursă unde pot exista variabile substituabile este numită *zonă substituabilă*. Prin *substituție* se va înțelege procesul de analiză a unei zone substituabile, delimitarea variabilelor substituabile și obținerea *zonei substituite*. Procesul de *substituție* are loc în felul următor:

— se caută în zona substituabilă (de la stînga spre dreapta) o primă variabilă substituabilă după care se copie în zona substituită porțiunea de șir aflată pînă la primul &;

— dacă variabila substituabilă este de forma:

$$\& < simbol > \text{ sau } \& < simbol >.$$

atunci simbolul este sau nume de indice sau nume de șir și zona substituită va fi completată, prin concatenare, cu șirul de caractere desemnat prin numele de șir sau cu valoarea zecimală a indicelui desemnat prin numele de indice (nu sînt posibile valori negative pentru indici);

— dacă variabila substituabilă este de forma

$$\& < simbol > (< expresie >)$$



Scanned with OKEN Scanner

atunci simbolul desemnează sau o funcție intrinsecă sau un șir cu structură de listă ; după caz se va completa zona substituită cu valoarea numerică a funcției sau cu elementul din listă specificat prin valoarea expresiei ;

— dacă variabila substituabilă este de forma

$$\& < simbol > (< expresie >, < expresie >)$$

atunci simbolul desemnează un *subșir* de lungime $l$ începînd cu al $n$-lea caracter dintr-un șir principal, unde $n$ și $l$ sînt respectiv valorile celor două expresii ;

— dacă zona substituabilă nu este epuizată, se continuă transferul de caractere în zona substituită pînă la întîlnirea unui nou & sau pînă la sfîrșitul zonei.

**Exemple de substituții.** Fie următoarele definiții de indici (I,J) și de șiruri (SIR1, SIR2) :

```
MINDEX      I=2
MINDEX      J=4
MSTRING     SIR1='ABCD', 'SIR2'
MSTRING     SIR2='MN', 'OPQ', 'RSTU', 'VWXYZ'
```

atunci fiecare din variabilele substituabile de mai jos se va substitui prin valoarea indicată în dreapta ei :

```
&I                  ..... 2
&I+ & J             ..... 6
&SIR1               ..... ABCDSIR2
&SIR1.(2)           ..... SIR2(2)
&SIR2(J— I)         ..... OPQ
&SIR2(J)            ..... VWXYZ
&MC(SIR1(1))        ..... 4
&(ML(SIR2)          ..... 4
&MN(SIR1(1))        ..... 0
&MS(& SIR1.(2))     ..... Ol Q
&SIR1(3,4)          ..... CLSI
```

7.1.3. **Instrucțiuni de generare.** În continuare se vor prezenta pe scurt instrucțiunile de generare din limbajul MAGIRIS.

a) *Definirea de șiruri — instrucțiunea* MSTRING

F o r m a t :

[< *etichetă* >] MSTRING < *nume-șir* ⊳ = < *listă-elemente* >

unde

< *listă-elemente* > : : = < *element* > | < *listă-elemente* >, < *element* >
< *element* > : : = < *constantă-șir* > | < *nume-șir* > |
< *nume-șir* > (< *expresie* >) |
< *nume-șir* > (< *expresie* >, < *expresie* >)



Scanned with OKEN Scanner

**Exemple**

```
MSTRING     SIR1='ABCD'
MSTRING     SIR2=SIR1, 'EF', 'GHIJ'
MSTRING     SIR3=SIR1(3,2), SIR2(2)
```

Aici simbolul SIR3 va desemna şirul CDEF (şir cu două elemente).

Observaţie. Şirurile definite prin MSTRING sînt memorate într-o zonă de dimensiune 2K, iar redefinirea unui şir antrenează recuperarea locului ocupat de vechea definire.

b) *Definirea de indici — instrucţiunea* MINDEX

Format:

[< *etichetă* >] MINDEX < *nume-indice* >=< *expresie* >

**Exemple**

```
MINDEX      I=120
MINDEX      K=I/6
MINDEX      J=J+1
```

Observaţii. 1. Valoarea afectată unui indice nu poate depăşi $2^{23}-1$.
2. La un moment dat într-un prototip nu pot exista mau mult de 16 indici distincţi.

c) *Instrucţiunea* MNOP

MNOP este o instrucţiune inefectivă, folosită ca suport pentru etichete de generare.

Format:

[< *etichetă* >] MNOP

**Exemplu**

```
            MDO         100,I=1,10
            DATA,4,4      I
100         MNOP
```

d) *Definirea comentariilor-instrucţiunea* MNOTE

Format:

MNOTE < *comentariu* >

**Exemplu**

MNOTE MACROGENERARE PROGRAM P1

Observaţie. Cu MNOTE se definesc comentarii de generare. Aceste comentarii nu sînt incluse în textul obiect generat.

e) *Memorarea textului sursă-instrucţiunea* MARK

Format:

[< *etichetă* >] MARK < *eticheta-1* >

Observaţie. Deoarece textul sursă este tratat într-un singur pas, fără posibilitate de citire înapoi, salturile în amonte nu sînt posibile decît după înregistrarea textului sursă în memorie, într-o zonă special rezervată. Instrucţiunea MARK deter-



Scanned with OKEN Scanner

mină stocarea tuturor liniilor sursă care o urmează imediat pînă la instrucțiunea cu eticheta < *eticheta*-1 >. Dimensiunea zonei de stocare este de 1 440 locații. Prima instrucțiune este chiar MARK.

f) *Salt necondiționat - instrucțiunea* MGO

Format:

[< *etichetă* >] MGO < *eticheta*-1>

Observații. Instrucțiunea MGO în acest format permite dirijarea necondiționată a generării la instrucțiunea MAGIRIS cu eticheta < *eticheta*-1 >. Dacă instrucțiunea referită este după MGO atunci saltul este efectiv întotdeauna ; în caz contrar saltul nu este posibil decît dacă textul a fost memorat printr-o instrucțiune MARK sau MDO.

g) *Salt condiționat - instrucțiunea* MGO

Format:

[< *etichetă* >] MGO <*eticheta*-1 >, < *zonă-substituabilă* >

După operația de substituție zona substituabilă trebuie să devină o expresie relațională. Saltul la instrucțiunea MAGIRIS ce are eticheta < *eticheta*-1> se produce efectiv numai dacă expresia relațională are valoarea 1 (adevărat) ; în caz contrar se continuă cu instrucțiunea ce urmează imediat după MGO. Evident este necesară stocarea textului sursă dacă saltul este în amonte.

În exemplul următor saltul va avea loc la instrucțiunea cu eticheta 100 atît timp cît $I \leqslant 10$.

```
        MINDEX      I=1
        MARK        200
100     MNOP
        DATA,1,4    &I
        MINDEX      I=I+1
200     MGO         100,I,LE,10
```

h) *Salt dirijat – instrucțiunea* MGO

Format:

< *etichetă* > MGO (< *listă-etichete* >) < *expresie* >

unde

< *listă-etichete* > : : = < *etichetă* > | < *listă-etichete* >, < *etichetă* >

Dacă lista conține $n$ etichete $e_1, e_2, \ldots, e_n$ și $k$ este valoarea expresiei, atunci saltul are loc la instrucțiunea $e_k$ dacă $1 \leqslant k \leqslant n$; dacă $k \notin \{1, 2, \ldots n\}$, atunci se continuă cu instrucțiunea ce urmează imediat după MGO. Expresia din instrucțiune poate fi obținută și prin substituție.



Scanned with OKEN Scanner

i) *Bucle de generare – instrucțiunea* MDO

Format:

[< *etichetă* >] MDO < *eticheta-1* >, < *zonă substituabilă* >

unde zona substituabilă după substituție trebuie să fie de una din formele :

< *nume-indice* >=< *v-inf* >, < *v-sup* >, < *pas* >

sau:

< *nume-indice* >=<*v-inf* >, < *v-sup* >

În această situație < *v-inf* >, < *v-sup* > și < *pas* > pot fi numere întregi fără semn sau nume de indici și dau valoarea inferioară, valoarea superioară a indicelui de ciclare și respectiv cantitatea cu care este mărită valoarea indicelui de ciclare la fiecare execuție a buclei. Instrucțiunea MDO marchează începutul unei bucle de generare, corpul buclei fiind format din instrucțiunile ce urmează imediat după MDO și pînă la instrucțiunea cu eticheta < *eticheta-1* > inclusiv. La întîlnirea lui MDO se declanșează stocarea corpului buclei făcînd inutilă utilizarea instrucțiunii MARK.

**Exemplu.** Secvența următoare :

```
        MDO         200,I=1,10,1
        DATA,1,4    &I
200     MNOP
```

va genera un text ASSIRIS identic cu cel generat de secvența prezentată ca exemplu la punctul g).

j) *Definirea parametrilor generării – instrucțiunea* MPAR.

Format:

< *etichetă* > MPAR < *zonă substituabilă* >

După substituție zona substituită trebuie să fie o listă de parametri :

< *lista parametri* > : : = < *definiție parametru* > |
< *listă parametri* >, < *definiție parametru* >

< *definiție parametru* > : : = MGO = < *constantă numerică* > |
MLAB = < *constantă numerică* > |
MCOM = < *constantă numerică* > |
MARG = < *constantă numerică* >

Această instrucțiune permite atribuirea de valori pentru cel puțin unu din parametrii generării : MGO, MLAB, MCOM, MARG. Semnificația parametrilor este următoarea :

MGO definește numărul maxim de salturi care pot fi efectuate în cazul în care dintr-o eroare s-a programat prin MGO (salt în amonte) sau MDO o buclă infinită.



Scanned with OKEN Scanner

MLAB permite definirea numărului maxim de etichete după care se reia generarea în cazul în care printr-o instrucțiune MARK, MDO, MGO (salt înainte) s-a făcut referire la o instrucțiune cu etichetă eronată de exemplu. Se evită astfel parcurgerea întregului text sursă fără posibilitate de generare.

MCOM definește numărul coloanei de origine a părții de comandă a instrucțiunilor generate.

MARG definește numărul coloanei de origine a părții de argumente a instrucțiunilor generate.

De exemplu, dacă instrucțiunile generate sînt în limbaj ASSIRIS, se va lua MCOM=12 și MARG=22. Dacă parametrii sînt omiși, se consideră MCOM=10, MARG=22, MLAB=21, MGO=1 024, iar dacă MCOM=0 se suprimă cadrajul textului generat.

k) *Apelarea unui prototip — instrucțiunea* MFETCH.

Instrucțiunea MFETCH permite apelarea, într-un program MAGIRIS, a unui prototip catalogat anterior într-o bibliotecă sursă și definirea parametrilor acestuia.

F o r m a t :

[< *etichetă* >] MFETCH < *definire prototip* >[, < *definire parametri* >]

unde :

< *definire prototip* > : : = LN : < *nume bibliotecă* >,
FN : < *nume prototip* >,
GN : < *generație bibliotecă* >,
VN : < *versiune bibliotecă* >,
DV : < *adresă disc* >,
UN : < *număr actualizare prototip* >

< *definire parametri* > : : = [< *listă definiri indici* >] [, < *definire șir* >]

< *listă definiri indici* > : : = < *definire indice* > |
< *definire indice* >, < *listă definiri indici* >

< *definire indice* > : : = < *nume indice* > = < *expresie* >

< *definire șir* > : : = < *nume șir* > = < *listă elemente* >

**Exemplu**

MFETCH LN : BIBL1,FN : PROTOTIP,GN : 1,VN : 0,UN:1,A='ALFA'

1) *Sfîrșit text sursă — instrucțiunea* MEND

F o r m a t :

[< *etichetă* >] MEND

Este o instrucțiune obligatorie.



Scanned with OKEN Scanner

## 7.2. LIMBAJUL LPS

Macrogeneratorul LPS este un translator independent de compilatorul LPS. Textul sursă pentru macrogeneratorul LPS este în general compus din instrucțiuni ale limbajului LPS, apeluri de macroinstrucțiuni, metainstrucțiuni (instrucțiuni de generare) și din definiții de macroinstrucțiuni (modele sau prototipuri). Definițiile de macroinstrucțiuni pot eventual să fie în prealabil catalogate în biblioteci sursă. La rîndul său textul obiect generat poate fi catalogat într-o bibliotecă sursă și poate fi într-un limbaj de programare oarecare (de exemplu LPS, ASSIRIS, FORTRAN, COBOL etc.). Dacă textul generat este în limbaj LPS, atunci el poate fi imediat compilat, textul obiect generat (în format RBN) fiind memorat pe discul sistem în fișierul FILEDIT și eventual catalogat într-o bibliotecă RBN.

În cele ce urmează se vor prezenta pe scurt cîteva elemente privind definirea și apelul de macroinstrucțiuni LPS. După cum rezultă și din exemplul următor (fig. 7.2) o definire de macroinstrucțiune presupune : definirea tipului acesteia, definirea modului de apel, definirea parametrilor formali, definirea unor macrovariabile (variabile utilizate în cursul procesului de generare) și definirea procesului de generare. Macroinstrucțiunea

```
    MACROS ;                                    DECLARARE TIP MACRO
<TABP5>  = @P ;                                 DEFINIRE FORMA APEL MACRO
 @P      = NR ;                                 DEFINIRE PARAM FORMAL @P
    MSTART ;
      MDCL LOCAL @I  INTEGER ;                  DECLARARE MACRO VARIABILE
      MDCL LOCAL @P5 INTEGER ;                  LOCALE  @I SI @P5
      MSET  @P5 = 1 ;
      MGEN  ('TABP5  RES,4       0') ;
      MDO   .DOM @I=1,@P,1 ;                     DESCRIEREA PROCESULUI
        MSET  @P5=@P5*5 ;                           DE GENERARE
        MGEN  ('P' @I '      DATA,4,4      @P5);
.DOM:MNOP
   MEND                                         SFIRSIT TEXT MACRO
```

Fig. 7.2. Definire de macrostrucțiune LPS.

dată ca exemplu permite generarea în limbaj ASSIRIS a unei tabele cu primele *n* puteri ale lui 5. Aici *n* este valoarea care se transmite în momentul apelării macroinstrucțiunii. De exemplu dacă $n=3$, atunci apelul este de forma

$$< \text{TABP5} \quad 3 >$$

iar textul generat (sau expandarea macroinstrucțiunii) va fi

```
TABP5  RES,4       0
P1     DATA,4,4    5
P2     DATA,4,4    25
P3     DATA,4,4    125
```



Scanned with OKEN Scanner

7.2.1. **Tipuri de macroinstrucţiune LPS.** În funcţie de modul de definire a sintaxei apelului există două tipuri de macroinstrucţiuni :

a) *macroinstrucţiuni de tip standard* cu apel poziţional (parametrii formali şi actuali se asociază prin rangul lor în instrucţiunile de definiţie şi de apel), cu apel pe bază de cuvinte cheie (parametrii se asociază pe bază de cuvinte cheie) şi cu apel mixt (combinaţie a celor două tipuri) ;

b) *macroinstrucţiuni de tip sintactic* care prin sintaxa apelului permit de exemplu un număr variabil de parametri, inserarea de şiruri de caractere în apel (deci o definire mai clară a apelului) etc.

Prima linie cu care începe orice definiţie de macroinstrucţiune este un indicator de tip, care defineşte tipul macroinstrucţiunii, urmat de caracterul „;". Indicatorul de tip poate fi :

| Indicator de tip | Tip macroinstrucţiune |
|---|---|
| MACROS | Macroinstrucţiuni standard cu apel poziţional. |
| MACROK | Macroinstrucţiuni standard cu apel pe bază de cuvinte cheie. |
| MACROM | Macroinstrucţiuni standard cu apel mixt. |
| MACRO | Macroinstrucţiuni de tip sintactic. |

7.2.2. **Definirea modului de apel.** Nu se va prezenta decît pentru cazul macroinstrucţiunilor standard. În această situaţie definirea apelului este de forma

$$< \textit{nume-macro} > \; = \; \textit{listă-parametri-formali} \; ;$$

Se remarcă faptul că numele macroinstrucţiunii (care este o succesiune de litere şi cifre, primul caracter fiind obligator o literă) este inclus între simbolurile < şi >. În listă parametrii formali sînt separaţi prin virgule, fiecare parametru fiind un identificator precedat de simbolul@

a) În cazul macroinstrucţiunilor standard cu apel poziţional parametrii formali formează o listă simplă, de exemplu :

$$< \text{MACRO1} > = @\text{P1}, @\text{P2}, @\text{P3} \; ;$$

unde MACRO1 este numele macroinstrucţiunii, iar @ P1, @ P2, @ P3 sînt parametri formali.

b) În cazul macroinstrucţiunilor standard cu apel pe bază de cuvinte cheie parametrii formali se pot scrie în listă sub una din formele :

*(cuvînt-cheie = valoare)*
*(cuvînt-cheie : valoare)*



Scanned with OKEN Scanner

*(cuvînt-cheie , valoare)*
*cuvînt-cheie = valoare*
*cuvînt-cheie : valoare*

unde *cuvînt-cheie* este un identificator precedat de@ , iar valoarea este opțională (dacă este prezentă, aceasta va înlocui parametrul formal în cazul în care parametrul actual corespunzător este omis din lista de apel. Corespondența între parametrii actuali și cei formali se stabilește pe bază de cuvinte cheie, deci ordinea parametrilor în lista de apel nu este importantă.

**Exemplu.** Fie definiția

< MACRO 1 > =(@ P1 :),(@ P2 : A),@ P3= ;

Dacă apelul este forma :

< MACRO1 (P1 : X), P3=17>

atunci valorile luate de parametri vor fi :

| | |
|---|---|
| @ P1 | X |
| @ P2 | A |
| @ P3 | 17. |

iar în cazul apelului :

< MACRO1 (P2:B), P3=20 >

valorile luate de parametri vor fi :

| | |
|---|---|
| @ P1 | șir vid |
| @ P2 | B |
| @ P3 | 20 |

c) În cazul macroinstrucțiunilor standard cu apel mixt, parametrii poziționali trebuie obligatoriu să preceadă în listă parametrii cu cuvinte-cheie. Exemplul următor (fig. 7.3) constituie o definiție de macroinstrucțiune de acest tip, aceasta permițînd generarea unei secvențe de instruc-

```
      MACROM ;
<GET>     =@FISIER,(RCA:),(RCS:);
 @FISIER = IDENT ;
 @RCA    = IDENT ;
 @RCS    = NB    ;
    MSTART ;
      MGEN ('    LD4I,2     ' @FISIER);
      MIF (@RCA ='') .IFM1 ;
      MGEN ('    LD4I,3     ' @RCA);
 IFM1:MIF (@RCS = '') .IFM2 ;
      MGEN ('    LD4I,4     ' @RCS);
 IFM2:MGEN ('    LDII,2    X''A1''');
      MGEN ('    BAL,8     *' @FISIER '+32');
    MEND ;
```

Fig. 7.3. Definiție de macroinstrucțiune LPS standard cu apel mixt.



Scanned with OKEN Scanner

ţiuni ASSIRIS corespunzînd macroinstrucţiunii GET din limbajul de gestiune a fişierelor. Parametrii sînt : @ FIŞIER (eticheta instrucţiunii de descriere a fişierului secvenţial), @ RCA (adresa zonei articol) şi @ RCS (lungimea zonei articol).

Dacă apelul este de forma

< GET FCART, (RCA : ZCART), (RCS : 50) >

atunci secvenţa generată va fi :

```
LD4I,2    FCART
LD4I,3    ZCART
LD4I,4    50
LD1I,2    X'A1'
BAL,8     *FCART+32
```

7.2.3. **Definirea parametrilor.** Se precizează pentru fiecare parametru formal tipul parametrului actual corespunzător în momentul apelului. De exemplu, o definire de forma

@ P1 = IDENT | STRING

precizează că parametrul actual corespunzător lui @P1 poate fi un identificator sau un şir de caractere.

Parametrii actuali pot avea următoarele tipuri :

| Tip | Semnificaţie | Exemplu de reprezentare | Valoare asociată |
|---|---|---|---|
| IDENT | identificator | ALFA 1 | ALFA1 |
| ASTIDENT | *urmat de un identificator | ALFA | *ALFA |
| STRING | şir de caractere | 'AB''C' | AB'C |
| L | literă | A | A |
| C | cifră | 7 | 7 |
| NB | număr natural | 173 | 173 |
| NBS | număr întreg | −78 | −78 |
| HEXA | număr hexazecimal | X'4A52' | 4A52 |

În absenţa declaraţiei de tip un parametru se consideră de tip IDENT

7.4.2. **Declararea macrovariabilelor.** În LPS se disting două clase de macrovariabile :

— *macrovariabile locale* care nu sînt utilizabile decît în macroinstrucţiunile unde sînt declarate ; acestea sînt iniţializate la fiecare apel al macroinstrucţiunii şi anulate la sfîrşit ;

— *macrovariabile globale* care permit comunicarea între macroinstrucţiuni şi textul sursă ; acestea sînt generate la începutul prelucrării textului sursă.



Scanned with OKEN Scanner

Indiferent însă de clasă macrovariabilele pot fi :

— *întregi*, reprezentate sub formă de cuvînt ;

— *booleene*, care nu pot lua decît două valori : adevărat și fals ;

— *de tip șir de caractere* pentru care trebuie precizat numărul de caractere.

O metainstrucțiune de declarare de macrovariabilă cuprinde :

— cuvîntul cheie MDCL ;

— indicatorul de clasă care poate fi LOCAL sau GLOBAL (prin absență se consideră GLOBAL) ;

— numele macrovariabilei (identificator precedat de @ ) ;

— indicatorul de tip care poate fi : INTEGER (prescurtat I) pentru macrovariabile întregi, BOOLEAN (prescurtat B) pentru macrovariabile booleene sau CHARACTER (prescurtat C) pentru macrovariabile de tip șir ; în acest caz trebuie imediat precizat numărul maxim de caractere sub formă de număr întreg pozitiv (maximum 255) între paranteze ;

— declararea eventuală a unei valori inițiale sub forma :

INITIAL = *valoare*

unde *valoare* se scrie funcție de tipul macrovariabilei și anume : un întreg cu semn pentru cele de tip I, .T. sau .F. pentru cele de tip B și un șir de caractere între apostrofuri pentru cele de tip C ;

— în final caracterul ; .

**Exemple**

```
MDCL    LOCAL     @ I      INTEGER ;
MDCL    @ S       BOOLEAN INITIAL=.F. ;
MDCL    LOCAL     @ SIR C(10) INITIAL='ALFA' ;
```

7.2.5. **Descrierea procesului de generare.** Procesul de generare este descris prin metainstrucțiuni de tipul MGEN, MIF, MDO, MSET etc. după cum rezultă-și din exemplele anterioare. Metainstrucțiunile pot fi etichetate, o etichetă fiind constituită dintr-un identificator precedat de caracterul punct. Separarea etichetei de restul frazei se face prin caracterul :. În cele ce urmează vor fi prezentate cîteva clase de metainstrucțiuni ce sînt utilizate pentru definirea procesului de generare.

a) *Metainstrucțiunile generative* pot avea două forme : o formă standard și o formă sintactică. În primul caz (forma standard) este posibil doar procesul de substituție a variabilelor substituabile (parametri formali și macrovariabile) cu valorile actuale. Metainstrucțiunea se prezintă ca o frază cu structură oarecare, sfîrșitul acesteia fiind delimitat prin caracterul ; , iar variabilele substituabile fiind reperate prin caracterul@ . Exemplul următor (fig. 7.4) prezintă o macroinstrucțiune standard cu apel pozițional ce permite generarea unei subrutine de tip FUNCTION, în limbaj FORTRAN, pentru determinarea produsului scalar a doi vectori



Scanned with OKEN Scanner

```
    MACROS;
<PSCALAR> = @NUMEF,@DIM ;
 @NUMEF   = IDENT ;
 @DIM     = NB(IDENT ;
    MSTART ;
C  TEXTUL FUNCTIEI DE CALCUL
C    AL PRODUSULUI SCALAR
       FUNCTION @NUMEF (T1,T2)
       DIMENSION T1(@DIM),T2(@DIM)
       @NUMEF=0
       DO 100 I=1,@DIM
         @NUMEF=@NUMEF+T1(I)*T2(I)
  100  CONTINUE
       RETURN
       END
C  SFIPSIT TEXT FUNCTIE ;
    MEND ;
```

Fig. 7.4. Definiție de macroinstrucțiune LPS standard cu apel pozițional

T1 și T2, avînd drept parametri numele funcției (@NUMEF) și dimensiunea celor doi vectori (@DIM). Se remarcă faptul că tot textul subrutinei este considerat ca o metainstrucțiune generativă.

În al doilea caz (forma sintactică de generare) este posibilă generarea unui text conform unei anumite structuri care se descrie printr-o metainstrucțiune MGEN, care are formatul

MGEN (*listă*) ;

unde *lista* descrie modul cum se face generarea, ea fiind o succesiune de parametri formali, macrovariabile, funcții intrinseci, șiruri de caractere (de exemplu 'ALFA') etc. Interpretarea unei metainstrucțiuni determină generarea unei linii de text în limbajul obiect, la sfîrșit fiind inserat cel puțin un caracter „blanc" (vezi fig. 7.1 și 7.2).

b) *Metainstrucțiunile pentru controlul generării* descriu operații de salt necondiționat (MGO), salt condiționat (MIF), operație inefectivă (MNOP), bucle de generare (MDO și MWHILE) terminare necondiționată a generării (MEXIT).

*Salt necondiționat* se scrie sub forma

MGO *etichetă*;

*Salt condiționat.* Se scrie sub una din formele :

MIF (*expresie-booleană*) *etichetă* ;

MIF (*expresie-booleană*) MEXIT ;

Dacă *expresia booleană* are valoarea adevărat, atunci în primul caz se execută salt la metainstrucțiunea cu eticheta *eticheta*, iar în al doilea caz se termină necondiționat generarea (vezi MEXIT) ; dacă expresia are valoarea fals, atunci interpretarea continuă în secvență.

*Operație inefectivă* se scrie sub forma

MNOP ;

și permite de a defini etichete necesare pentru delimitarea corpului buclelor.



Scanned with OKEN Scanner

*Bucle de generare tip* MDO se scriu sub forma

MDO *et* @ *var* = $n_1$, $n_2$, $n_3$ :

corp buclă

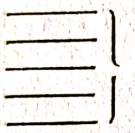
*et* : M NOP ;

unde *et* este o etichetă, @ *var* o macrovariabilă de tip INTEGER (variabila de control al buclei), iar $n_1$, $n_2$, $n_3$ sînt numere naturale sau macrovariabile de tip INTEGER care dau valoarea inițială și finală a variabilei de control a buclei și respectiv pasul. Macrovariabila @ *var* este inițializată cu valoarea $n_1$ și mărită la fiecare repetare a buclei cu $n_3$ pînă cînd devine mai mare ca $n_2$.

Se pot crea structuri cu bucle incluse una în alta în număr nelimitat, fiecare buclă fiind definită de un MDO și un MNOP.

*Bucle de generare tip* MWHILE se scriu sub forma

MWHILE *et* (*expresie-booleană*) ;



corp buclă

*et* : MNOP ;

Corpul buclei este interpretat repetitiv atît timp cît la începutul reluării buclei *expresia booleană* are valoarea adevărat.

*Terminarea necondiționată* se scrie sub forma

MEXIT ;

Această metainstrucțiune are o acțiune care depinde de contextul în care apare. Astfel, dacă apare într-o macroinstrucțiune, atunci ea determină sfîrșitul interpretării acesteia, în timp ce dacă apare în textul sursă, indică sfîrșitul macrogenerării.

c) *Metainstrucțiunile de calcul* sînt de forma

MSET @ *var* = *expresie* ;

unde @ *var* este o macrovariabilă, iar *expresie* este o expresie *aritmetică, booleană, de tip șir de caractere* sau o *funcție intrinsecă.*

*Expresiile aritmetice* sînt compuse dintr-un termen sau din doi termeni uniți prin unul din operatorii :

\+ (adunare), − (scădere), * (înmulțire), / (împărțire).

La rîndul său un *termen* poate fi un număr natural, o macrovariabilă de tip INTEGER sau o funcție intrinsecă de tip INTEGER.

**Exemple**

```
MSET@S=@A+2 ;
MSET@P1=@P2*@P3 ;
```

unde @S, @P1 pot fi macrovariabile de tip I sau C, iar @A, @P2, @P3 de tip I.

*O expresie booleană* poate fi sau o *condiție* sau o *expresie relațională.*

O *condiție* se compune dintr-un *termen boolean* sau din doi *termeni booleeni* separați prin unul din operatorii : | (sau logic), & (și logic). Primul termen poate fi precedat de operatorul # (negare logică). Un *termen boolean* la rîndul său poate fi o macrovariabilă de tip BOOLEAN sau una din valorile logice : .T. (adevărat) sau .F. (fals).

O *expresie relațională* se compune din doi *termeni relaționali* de același tip, separați prin unul din operatorii :

| | |
|---|---|
| < (mai mic), | <=(mai mic sau egal), |
| > (mai mare), | >=(mai mare sau egal), |
| = (egal), | # =(diferit de). |

Primul dintre termeni poate fi precedat de unul din semnele +, — sau #. Un *termen relațional* poate fi o macrovariabilă de orice tip, un parametru formal, o valoare de orice tip și o funcție intrinsecă de orice tip. În funcție de tipul termenilor compararea poate fi :

— aritmetică, dacă termenii sînt de tip I ;

— logică, dacă termenii sînt de tip B, admițînd că .T.>.F. ;

— între șiruri dacă termenii sînt de tip C admițînd că șirurile sînt cadrate la stînga și reprezentate în cod EBCDIC.

**Exemple**

```
MSET@B=@A&.T. ;
MSET@B1=@D<=100 ;
```

unde @B, @B1, @A sînt macrovariabile de tip B, iar @D de tip I.

O *expresie de tip șir* se compune dintr-un termen de tip șir sau din doi termeni de tip șir uniți prin operatorul de concatenare notat . CONC. Un termen de tip șir poate fi o macrovariabilă de tip C, un parametru formal, un șir de caractere sau o funcție intrinsecă de tip C.

**Exemplu.** Dacă

```
MDCL@A      C(9) ;
MDCL@B      C(5)    INITIAL 'ABCDE' ;
```



Scanned with OKEN Scanner

atunci

MSET@A=@B.CONC.'FGHI' :

va determina ca@A să ia valoarea 'ABCDEFGHI'.

Observații. 1) Nu este permisă modificarea valorii unui parametru formal.
2) Parametrii se consideră de tip CHARACTER.

*Funcții intrinseci* pot fi :

— funcția L ( @ *var*) de tip INTEGER, al cărei rezultat este lungimea șirului de caractere desemnat prin @ *var* ;

— funcția T (@ *var*) de tip CHARACTER al cărei rezultat este tipul șirului de caractere desemnat prin @ *var* și anume :

| Valoare T | Tip șir |
|---|---|
| N | număr fără semn |
| Z | număr cu semn |
| X | număr hexazecimal |
| D | identificator |
| S | șir de caractere |
| A | asterisc urmat de identificator |
| U | fără o structură particulară |

— funcția CHAR ( @ *var*, $n_1$, $n_2$) care permite de a substitui doar o parte a șirului desemnat de @ *var* și anume cea care începe cu caracterul de rang $n_1$ (rangul primului caracter este zero) și are lungimea de $n_2$ caractere ; $n_1$ și $n_2$ sînt numere naturale.

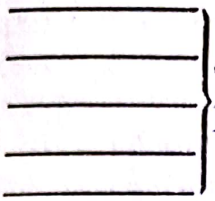
7.2.6. **Structura unui program în limbaj** MACRO-LPS. Schematic ea este similară cu cea prezentată în exemplul următor :

```
. COMPILE LPS,MACRO
* MFILE MU(F),DV:AD1,VN:0,GN:1,LN:BIBMACRO,
  FN:REPMACRO
  MFETCHS MU ;
_____
_____
_____
_____
_____
```

} Definiții de macroinstrucțiuni de tip MACROS, MACROM, MACROK, MACRO



Scanned with OKEN Scanner




- EOF

unde MFILE descrie o bibliotecă de tip sursă (BIBMACRO) utilizată exclusiv pentru definiții de macroinstrucțiuni (parametrul F). Metainstrucțiunea MFETGHS declară că toate definițiile de macroinstrucțiuni, care urmează după ea în program, vor fi incluse în biblioteca descrisă prin MFILE avînd idexul MU (idex de asociere asemănător cu identificatorul de exploatare la fișiere). REPMACRO este numele fișierului de macroinstrucțiuni din bibliotecă unde vor fi incluse macroinstrucțiunile definite actualmente în program.

Dacă se dorește catalogarea textului obiect generat, atunci în cartela COMPILE se adaugă parametrii de descriere ai bibliotecii. În cazul în care textul generat este în limbaj LPS și se dorește compilarea sa, cartela COMPILE se modifică sub forma :

- COMPILE LPS, MACRO, LPS

Scanned with OKEN Scanner

# 8

# LIMBAJE CONVERSAȚIONALE

În acest capitol se prezintă două limbaje concepute pentru a fi utilizate în sisteme *on-line* și destinate în principal pentru rezolvarea de probleme tehnico-științifice. Asemenea limbaje au fost definite în așa fel încît utilizatorii situați la terminale conversaționale să poată efectua ușor diverse operații de calcul. Au fost definite și implementate multe asemenea limbaje: JOSS, BASIC, CAL, QUIKTRAN, AMTRAN, DIALOG, APL, TTPL etc.

## 8.1. LIMBAJUL BASIC

Limbajul BASIC (*Beginners All Purpose Symbolic Instruction Code*) [59, 60, 73, 74, 75] a fost creat la colegiul din Dartmouth în anul 1965, fiind implementat mai întîi pe un calculator de tip G.E. 225. Deși a fost conceput pentru a fi utilizat în regim conversațional, atît datele cît și programele fiind introduse și executate interactiv de la un terminal, limbajul BASIC poate fi ușor utilizat și în regim neconversațional în care caz programul și datele sînt perforate pe cartele și transmise spre prelucrare. Există sisteme de calcul care permit utilizarea limbajului BASIC în ambele moduri (inclusiv FELIX C-256 [113]).

8.1.1. **Elemente de bază ale limbajului** BASIC. Diversele implementări existente pentru limbajul BASIC diferă foarte puțin una de alta [60]. Limbajul are la bază un alfabet compus din *litere, cifre* și *caractere speciale*: *virgulă*, ;, +, −, *, /, ↑, =, paranteze și apostroful. Spațiile pot fi folosite de asemenea, sînt însă neglijate indiferent de locul în care apar în instrucțiunile BASIC. Un program BASIC constă dintr-o succesiune de linii, fiecare linie fiind compusă dintr-un *număr de linie* (un număr întreg)



Scanned with OKEN Scanner

urmat de o instrucțiune BASIC. În cadrul unui program instrucțiunile trebuie să fie ordonate crescător după numerele de linie și sînt, de obicei, executate una cîte una în această ordine.

*Instrucțiunile* BASIC încep cu un cuvînt cheie, ce indică tipul instrucțiunii, după care urmează o listă de argumente. De exemplu, comentariile pot fi inserate în programe prin instrucțiuni de forma :

< *număr-linie* > REM < *șir caractere* >

*Datele* prelucrate într-un program BASIC pot fi de două tipuri : numerice și șiruri de caractere. Datele numerice sînt reprezentate întotdeauna în calculator în virgulă mobilă. În cadrul unui program datele pot fi introduse sub formă de constante sau ca valori ale unor variabile.

*Constantele numerice* sînt scrise, obișnuit, cu sau fără punct zecimal și sînt precedate, eventual, de semn. Ele pot avea de asemenea un *factor de scală* (un întreg pozitiv sau negativ) precedat de litera E.

**Exemple**

0 ; −42 ; +003 ; −42.7 ; 35.72 : 7.75E20 ; 22E−3

(7.75E20 înseamnă $7{,}75 \times 10^{20}$ și 22E−3 înseamnă $22 \times 10^{-3}$).

*Constantele șir* se includ între apostrofuri.

*Variabilele* reprezintă valori care se pot modifica în timpul execuției unui program. În programe ele se reprezintă prin *nume de variabile* sau *identificatori*. Un nume de variabilă numerică poate fi o literă (A−Z) sau o literă urmată de o cifră (0−9), de exemplu A ; X ; F2 ; Z0 ; N etc. În schimb un nume de variabilă de tip șir constă dintr-o literă urmată de caracterul $, de exemplu A$ ; C$ ; Y$.

*Expresiile* BASIC sînt compuse din variabile, constante, operator aritmetici și paranteze. Ca operatori aritmetici se folosesc :

| | | |
|---|---|---|
| ** sau | ↑ | pentru ridicare la putere |
| * | | pentru înmulțire |
| / | | pentru împărțire |
| + | | pentru adunare |
| − | | pentru scădere |

În alcătuirea expresiilor trebuie să se țină seama de următoarele reguli :

1) doi operatori nu pot fi adiacenți ;
2) variabilele și constantele nu pot fi adiacente ;
3) orice expresie corectă poate fi inclusă între paranteze sau poate fi precedată de un semn ;
4) ierarhia dintre operatori este următoarea :

** sau ↑ , * sau /, + sau −.

*Instrucțiunile de atribuire* pentru variabilele numerice au formatul

< *număr-linie* > LET < *variabilă numerică* > = < *expresie* >

iar pentru variabilele de tip șir :

< *număr-linie* > LET < *variabilă șir* > = < *șir* >

unde

< *șir* > : : = < *variabilă șir* > | < *constantă șir* >

**Exemple**

```
100 LET Z =(X1-Y↑2)*(Z-T/ (T+S1))
200 LET B =20+0.0004*N
300 LET S =(X+R*R)/(2*R)
400 LET X$=D$
500 LET Y$='ALFA'
```

8.1.2. **Instrucțiuni de control.** În mod obișnuit instrucțiunile sînt executate în ordinea numărului de linie. Instrucțiunile de control dau o altă ordine de execuție.

*Instrucțiunea* G∅ T∅ transferă necondiționat controlul la altă instrucțiune al cărei număr se indică. Ea are forma

< *număr linie* > G∅ T∅ < *număr linie* >.

Ca și în alte limbaje există și o instrucțiune de tip G∅ T∅ calculat. Ea are forma

< *număr linie* > ON < *expresie* > G∅ T∅ < *listă numere linii* >

unde *lista de numere de linii* este o secvență de numere de instrucțiuni, separate prin virgulă. Selectarea numărului liniei se face în funcție de valoarea expresiei de după ON.

**Exemple**

```
10 ON I G0 T0 100, 200, 300, 400
```

*Instrucțiunea* IF *logic* are forma

< *număr linie* > IF < *expresie* > < *operator relațional* > < *expresie* > THEN < *număr linie* >

unde

< *operator relațional* > : : = < | < = | = | > | > = | < >

Operatorul „<>" semnifică „diferit de" sau „neegal cu". Dacă relația este îndeplinită, atunci controlul se transferă la instrucțiunea al cărei număr se află după THEN. În caz contrar se execută instrucțiunea ce urmează imediat după IF.



Scanned with OKEN Scanner

**Exemple**

400 IF N <=40000 THEN 500

120 IF N5 < > N6 THEN 50

*Instrucțiunile* FOR *și* NEXT. În general, orice ciclu se poate programa utilizînd instrucțiunile IF și GØ TØ. Totuși instrucțiunile FOR și NEXT constituie o metodă mai elegantă și mai economică de programare a structurilor repetitive (cicluri). Acestea au formatele :

< *număr linie* > FOR < *v* > = < *e-1* > TO < *e-2* > STEP < *e-3* >

< *număr linie* > NEXT < *v* >

unde < *v* > este o variabilă de control al ciclului, iar < *e-1* > < *e-2* > și < *e*-3 > sînt expresii (cu valori reale) reprezentînd respectiv valoarea inițială, valoarea finală a variabilei de control și pasul ciclului. Dacă pasul este omis, se presupune implicit 1.

Corpul ciclului este format din instrucțiunile BASIC cuprinse într-o instrucțiune FOR și una NEXT care citează aceeași variabilă de control <*v*>. În continuare se prezintă un exemplu de utilizare a instrucțiunilor FOR și NEXT (program 8.1).

```
10 FOR A=1 TO 5
20 PRINT "A=";A
30 FOR B=7.6 TO 8 STEP 0.2
40 PRINT TAB(5);"B=";B
50 FOR C=1 TO -1 STEP -1
60 PRINT TAB(10);"C=";C
70 NEXT C
80 NEXT B
90 NEXT A
92 END
```

Programul 8.1

Observații. Corpul unui ciclu poate :

a) să nu fie executat niciodată dacă valoarea inițială a variabilei de control este mai mare ca valoarea finală și pasul este pozitiv ;

b) să fie executat o dată dacă valoarile inițială și finală ale variabilei de control sînt identice ;

c) să fie executat de un număr infinit de ori dacă valoarea pasului este nulă (bineînțeles, dacă nu există în corpul ciclului o instrucțiune IF sau GØ TØ care să asigure ieșirea în afara lui).

*Instrucțiunea* STOP permite oprirea execuției programului în orice punct al acestuia, transmițînd controlul sistemului.

*Instrucțiunea* END trebuie să apară ca ultimă instrucțiune în orice program BASIC, avînd același efect ca și STOP.

8.1.3. **Introducerea și extragerea datelor.** *Instrucțiunea* PRINT se folosește pentru extragerea datelor la un terminal. Formatul instrucțiunii este

< *număr linie* > PRINT < *listă elemente* >

Lista cuprinde elemente ale căror valori trebuie tipărite sau afișate. Astfel de elemente pot fi : variabile, constante, expresii. Elementele listei pot fi separate prin virgulă și/sau punct și virgulă. Dacă elementele listei sînt separate prin virgulă, se consideră că o linie la terminal este compusă



Scanned with OKEN Scanner

din cinci zone a cîte 13 sau 15 caractere fiecare. Fiecare element al listei este afișat într-o zonă (sau mai multe dacă este necesar). Dacă instrucțiunea PRINT cuprinde mai mult de cinci elemente, afișarea celor ce nu au loc în prima linie se face pe următoarea. Aceste zone pot fi ignorate dacă virgula se înlocuiește cu caracterul ;. În acest caz fiecare element al listei este tipărit cu spațiu, înainte și după, suficient pentru a-l putea separa de cele din jur, cu excepția variabilelor și constantelor șir unde nu se lasă nici un spațiu.

Multe versiuni BASIC includ funcția TAB care poate face parte dintr-o listă PRINT. Ea are formatul.

TAB (< *expresie întreagă* >)

și permite deplasarea capului de scriere (sau a marcherului) cu un număr de poziții caracter egal cu valoarea expresiei de după TAB.

*Instrucțiunile* READ și DATA. Instrucțiunea READ permite atribuirea de valori pentru variabile șir și/sau numerice, valori ce sînt definite sub formă de constante în instrucțiuni DATA. Ele au formatele

< *număr linie* > DATA < *lista constante* >

< *număr linie* > READ < *listă variabile* >

Constantele și variabilele în liste se separă prin virgule. În timpul citirii programului BASIC datele introduse prin DATA sînt memorate într-o zonă de date exact în ordinea în care apar. În aceeași ordine sînt apoi atribuite variabilelor din READ.

De exemplu, la sfîrșitul secvenței de instrucțiuni (program 8.2) valorile variabilelor vor fi :

Q=1, R=5, S$=ALFA, T=70, U=45, A=−18, B=25, C=−81, D=50

Dacă în instrucțiunile READ din program există mai multe variabile decît date definite prin DATA, atunci se tipărește un mesaj la terminal.

Dacă într-un program este necesară prelucrarea unor date de mai multe ori, atunci cu ajutorul instrucțiunii RESTORE se poate relua citirea zonei de date de la capăt.

De exemplu, în programul 8.3 după execuție

A=−28.20, B=17.8, C=−28.20, C=19.4

```
90 DIM S$[4]
100 DATA 1,5,"ALFA",70
110 READ Q
120 DATA 45,-18,25
130 READ R,S$[1,4],T,U
140 READ A,B,C,D
150 DATA -81,50
```

Programul 8.2

```
100 DATA -28.2,17.8
110 READ A,B
120 RESTORE
130 READ C,D
140 PRINT A,B
150 PRINT C,D
999 END
```

Programul 8.3



Scanned with OKEN Scanner

*Instrucțiunea* INPUT are format similar cu READ :

< *număr linie* > INPUT < *listă variabile* >

În regim conversațional instrucțiunea INPUT determină oprirea execuției programului pentru ca valorile ce vor fi asociate variabilelor din listă să fie introduse de la terminal. În regim neconversațional datele pentru instrucțiunea INPUT se citesc de pe cartele. Indiferent însă de regimul de lucru datele de citit sînt numere și/sau șiruri de caractere separate prin virgule. Deoarece datele sînt separate prin virgule, utilizarea acestui caracter în șiruri este interzisă.

**8.1.4. Variabile indexate. Tablouri.** În BASIC se pot declara tablouri cu una sau două dimensiuni. Pentru aceasta se folosește instrucțiunea DIM ce are formatul :

< *număr linie* > DIM < *listă declarații tablouri* >

O declarație de tablou constă dintr-un nume de tablou (nume de variabilă format dintr-o singură literă în cazul tablourilor ale căror elemente sînt numere) urmat, între paranteze, de dimensiunea (sau dimensiunile) tabloului date sub formă de constante întregi.

**Exemplu**

```
100 DIM A (25),B (4,5),E (7,15),A$ (15)
```

Referiri la elementele tablourilor declarate se fac ca și în limbajul FORTRAN. De exemplu :

A(19), A(V), A(V+5), A(R*E), A$(10),
B(2,3), E(S,T), B(I−1, I*J)

Exemplul din programul 8.4 asigură inițializarea tabloului D(3,5).

O b s e r v a ț i e. Dacă numărul de elemente ale unui tablou este mai mic ca 10, atunci nu mai este necesară declararea lui prin DIM. Următorul exemplu (programul 8.5) calculează media aritmetică a patru numere introduse dinamic de la terminal.

```
100 DATA 11,12,13,14,15
200 DATA 21,22,23,24,25
300 DATA 31,32,33,34,35
400 DIM D[3,5]
500 REM  I-INDICE DE LINIE
700 FOR I=1 TO 3
800 FOR J=1 TO 5
900 READ D[I,J]
1000 NEXT J
1100 NEXT I
1300 MAT PRINT D
9999 END
```

Programul 8.4

```
100 LET M=0
110 FOR I=1 TO 4
120 INPUT N[I]
130 LET M=M+N[I]
140 NEXT I
150 PRINT "MEDIA ESTE",M/4
160 PRINT "NUMERELE SINT:"
170 FOR I=1 TO 4
180 PRINT N[I]
190 NEXT I
999 END
```

Programul 8.5



Scanned with OKEN Scanner

Majoritatea versiunilor limbajului BASIC au instrucțiuni pentru efectuarea de operații cu matrice (tablouri bidimensionale), instrucțiuni care nu se regăsesc în limbaje uzuale ca FORTRAN sau ALGOL 60. Aceste instrucțiuni permit evitarea utilizării instrucțiunii FOR, nu însă și a instrucțiunii DIM. Aceste instrucțiuni permit în general operații cu matrice de tipul celor ce sînt prezentate în exemplele care urmează :

```
100 DIM A(4,4), B(4,4), C(4,4)
```

| | |
|---|---|
| 200 MAT INPUT A,B | Introducerea linie cu linie de la terminal a elementelor matricelor *A* și *B*. |
| 300 MAT READ C | Citirea linie cu linie a elementelor matricei *C* din zona de date. |
| 400 MAT PRINT A,B | Tipărirea matricelor *A* și *B* linie cu linie. |
| 500 MAT A=B | Memorează matricea *B* în *A*. |
| 600 MAT C=A+B | Adună două matrice. |
| 700 MAT C=A−B | Scăderea a două matrice. |
| 800 MAT A=B*C | Înmulțirea a două matrice. |
| 900 MAT B=(*k*)*A | Înmulțirea tuturor elementelor matricei *A* cu constanta *k*. |
| 1010 MAT A=TRN (B) | Transpusa matricei *B* este memorată în *A*. |
| 1020 MAT A=INV (B) | Inversa matricei *B* este trecută în matricea *A*. |
| 1030 MAT A=ZER | Inițializarea cu zero a tuturor elementelor matricei *A*. |
| 1040 MAT B=CON | Inițializarea cu 1 a tuturor elementelor matricei *B*. |
| 1050 MAT C=IDN | Inițializarea cu 1 a elementelor matricei *C* de pe diagonală și cu zero a celorlalte ; *C* devine o matrice unitate. |

Observație. Operații de forma A=A+B sînt permise, în timp ce de forma A=A*B, A=INV (A) nu sînt permise. Exemplul din programul 8.6, permite determinarea soluției $(x_1, x_2)$ a unui sistem de două ecuații liniare cu două necunoscute, dată de relația $X=A^{-1}\cdot B$, unde

$$X=\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad A=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \text{și} \quad B=\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

```
10 REM MATRICEA A
20 DATA 15,75.2,17.75,-22.3
30 REM  MATRICEA  B
40 DATA 17,-22
50 DIM A[2,2],B[2,1],X[2,1],C[2,2]
60 MAT READ A,B
70 MAT C=INV(A)
80 MAT X=C*B
90 PRINT "SOLUTIA ESTE"
100 MAT PRINT X
999 END

SOLUTIA ESTE
-0.763972923

 0.378452046
```

Programul 8.6

8.1.5. **Funcții** BASIC. În BASIC există două tipuri de funcții : funcții intrinseci, și funcții definite de utilizator.

a) *Funcțiile intrinseci* sînt funcții des folosite ce au fost incluse ca elemente ale limbajului. Acestea sînt de forma

< *nume funcție* > ( < *argument funcție* >),

*argumentul funcției* fiind în general o expresie BASIC. Funcțiile intrinseci BASIC sînt prezentate în continuare (tabelul 8.1).



Scanned with OKEN Scanner

Tabelul 8.1

Funcții intrinseci în BASIC

| Funcția BASIC | Funcția matematică echivalentă |
|---|---|
| SIN (X) | sin ($x$) ; $x$ în radiani |
| COS (X) | cos ($x$) ; $x$ în radiani |
| TAN (X) | tg ($x$) ; $x$ în radiani |
| ATN (X) | arctg ($x$) ; arctangenta se obține în radiani |
| LOG (X) | ln $x$ |
| EXP (X) | $e^x$ |
| ABS (X) | $\|x\|$ |
| SQR (X) | $\sqrt{x}$ ; $x>0$ |
| SGN (X) | sign ($x$)=1 dacă $x>0$, 0 dacă $x=0$, —1 dacă $x<0$ |
| INT (X) | [$x$] ; partea întreagă a lui $x$ |
| RND (X) | Generator de numere aleatoare ; în mod obișnuit RND generează un număr cuprins între 0 și 1 |

Programul BASIC 8.7 calculează și afișează valoarea expresiei

$$A=\frac{1}{2}\lg\frac{1+\sin x}{1-\sin x}$$

pentru $x=0{,}25$ radiani.

b) *Funcțiile definite de utilizator* completează setul de funcții intrinseci într-un caz dat. Înainte de a fi utilizate în program acestea trebuie definite cu ajutorul instrucțiunii DEF :

< *număr linie* > DEF FN < *literă* > (< *parametru formal* >)= < *expresie* >

unde FN < *literă* > constituie numele funcției, parametrul formal este format numai dintr-o literă, iar expresia constituie definiția funcției folosind parametrul formal. Atunci cînd funcția este utilizată în program, parametrul formal este înlocuit printr-o variabilă actuală sau o expresie aritmetică.

În programul 8.8 se citesc trei unghiuri ale unui triunghi în grade și se tipăresc valorile sinusului lor. În acest caz A este parametru formal, iar atunci cînd funcția FNT este apelată, A este înlocuit pe rînd cu parametrii actuali $X(1)$, $X(2)$, $X(3)$.

```
100 DATA 0,25
200 READ X
300 LET X1=1+SIN(X)
400 LET X2=1-SIN(X)
500 LET N=2*LOG(10)
600 LET A=(LOG(X1/X2))/N
700 PRINT "A=";A
999 END

A= 0.109722594
```

Programul 8.7

```
10 REM  DEFINIRE FUNCTIE FNT(A)
20 DEF FNT(A)=SIN(A*0.017453)
30 FOR K=1 TO 3
40 READ X[K]
50 REM   APEL FUNCTIE FNT
60 LET S=FNT(X[K])
70 PRINT "SIN(";X[K];")=";S
80 NEXT K
90 DATA 120,35,25
99 END

SIN( 120  )= 0.866042954
SIN( 35   )= 0.573568050
SIN( 25   )= 0.422611634
```

Programul 8.8



Scanned with OKEN Scanner

```
10 DATA 25
20 READ N
30 REM  APEL SUBRUTINA
40 GOSUB 1000
50 PRINT "FACTORIAL DE";N;"=";F1
60 STOP
1000 REM  TEXT SUBRUTINA
1010 LET F1=N
1020 IF F1=1 THEN 1370
1030 FOR F2=F1-1 TO 1 STEP -1
1040 LET F1=F1*F2
1050 NEXT F2
1060 REM  SFIRSIT SUBRUTINA
1070 REM  RASPUNSUL IN F1
1080 RETURN
9999 END

FACTORIAL DE 25   = 1.55112E+25
```

Programul 8.9

8.1.6. **Subrutine în BASIC.** O subrutină BASIC se constituie ca o secvență obișnuită de instrucțiuni. Ea se apelează în program folosind instrucțiunea GOSUB :

< *număr linie* > GOSUB < *număr linie S* >

unde < *număr linie S* > reprezintă numărul primei linii din subrutină. Din subrutină controlul este transmis înapoi în programul apelant, la instrucțiunea ce urmează imediat după GOSUB, printr-o instrucțiune RETURN :

< *număr linie* > RETURN

Programul 8.9 prezintă o subrutină pentru calculul factorialului unui număr natural *N*.

## 8.2. LIMBAJUL APL

În forma originală limbajul APL a fost definit în 1962 [31].

O versiune modificată a limbajului APL, numită APL/360, a fost implementată imediat pe calculatoarele IBM 360. Ulterior au apărut și alte implementări, toate însă bazîndu-se pe versiunea APL/360 ([1, 28, 59, 60, 73, 75]).

Ca limbaj, APL are două caracteristici distinctive prin care se deosebește de alte limbaje și anume :

a) Este un limbaj interactiv, putînd fi utilizat de la terminale. Utilizatorul situat la terminal transmite un ordin. Calculatorul formulează răspunsul, îl transmite spre terminal și pregătește o nouă intrare. Acest ciclu se repetă pînă în momentul în care se anunță sfîrșitul dialogului. În APL/360 textul utilizatorului este aliniat după șase spații, în timp ce răspunsurile încep chiar de la marginea suportului. De exemplu :

```
      22.5+178.7   ← expresie utilizator
201.2              ← răspuns
      17+5×4
37
```

În continuare se poate introduce, de exemplu, o instrucțiune de atribuire, în care caz expresia din dreapta este evaluată, valoarea obținută fiind



Scanned with OKEN Scanner

atribuită variabilei dar nu și tipărită. Ca operator de atribuire se folosește simbolul←. De exemplu :

```
      A←17+5×4
      A
37
```

b) Se pot executa prelucrări cu structuri de date de tip tablou cu un număr oarecare de dimensiuni. Operațiile primitive definite în APL acceptă tablourile ca argumente și produc tablouri ca rezultat ; aceasta este o diferență esențială față de limbaje ca FORTRAN, ALGOL unde operațiile se execută cu elemente de tablouri.

8.2.1. **Elemente de bază ale limbajului** APL/360. Limbajul APL/360 are la bază un alfabet constînd din 26 litere (A – Z), zece cifre (0 – 9) și următoarele 52 caractere speciale :

+ − × ÷ * < ≤ = ≥ > ≠ ¯

. : , ; ¨ ( ) [ ] ? ! ' _ ∘

∨ ∧ ← ↓ → ↑ \ /

⊤ ⊥ | ∩ ∪ ⊃ ⊂

⌈ ⌈ ~ △ ▽ ○ □

α ω ∈ ρ ≀

**În afară de acestea se mai folosesc următoarele simboluri obținute prin supraimprimare de cîte două caractere speciale :**

⍋ ⍒ ⌽ ⍟ ⍫ ⍭ ! ⌹ ⌶ ⍉

Utilizarea caracterelor speciale ca operatori are avantajul că se obțin programe scurte și dezavantajul că programele obținute sînt așa de compacte încît ele sînt dificil de citit.

*Instrucțiunea* este unitatea de bază a oricărui program și este compusă dintr-o succesiune de operanzi și operatori sau separatori. În particular acestea pot conține doar un singur operator sau un singur operand.

*Operanzii* pot fi constante, variabile, sau operanzi elementari (scalari) și operanzi structurați (vectori, matrice, tablouri multidimensionale).

Un *nume de variabilă* constă dintr-o succesiune de litere și cifre, primul caracter fiind obligatoriu o literă.

*Operatorii* pot fi monadici sau diadici după cum operează cu un operand sau cu doi operanzi, din care cauză se mai numesc și funcții monadice sau funcții diadice. De asemenea operatorii se pot grupa și în : aritmetici,

logici, de relație, de acces, de atribuire, generatori, de intrare/ieșire, de salt. Menționăm că unii dintre operatori pot fi și monadici și diadici caz în care semnificația lor este complet diferită.

*Expresiile* APL sînt compuse din variabile, constante, operatori și paranteze. Între operatori nu este definită nici o prioritate atunci cînd expresia este evaluată, iar expresia este evaluată de la dreapta spre stînga. Parantezele sînt folosite în modul cunoscut de la expresiile matematice.

**8.2.2. Expresii scalare în APL.** Simplitatea sintaxei expresiilor scalare rezultă destul de clar din următoarea definiție :

$$\begin{aligned} &< \textit{expresie} > :: = < \textit{termen} > \mid \\ &\qquad < \textit{operator monadic} > \ < \textit{expresie} > \mid \\ &\qquad < \textit{termen} > \ < \textit{operator diadic} > \ < \textit{expresie} > \end{aligned}$$

$$< \textit{termen} > :: = < \textit{variabilă} > \mid < \textit{constantă} > \mid < \textit{expresie} >$$

Constantele din APL, ca reprezentare, sînt similare cu cele din BASIC cu singura observație că semnul minus este distinct de operatorul minus. $^{-}3$, $^{-}39.25$, $^{-}25.E6$, $^{-}2E$, $^{-}7$ sînt exemple de numere negative în APL, iar $X-3$ este o expresie ce conține operatorul diadic de scădere în timp ce $-X$ este o expresie ce conține operatorul monadic minus. Ca operatori aritmetici diadici se mai folosesc : $+$ pentru adunare, $\times$ pentru înmulțire, $\div$ pentru împărțire și $*$ pentru ridicare la putere. În expresii parantezele sînt folosite după convențiile obișnuite. Dacă parantezele nu sînt folosite, operațiile sînt executate secvențial de la dreapta spre stînga. Astfel, de exemplu, expresia $A \times B - C$, care în mod obișnuit este interpretată ca $(A \times B) - C$, în APL este interpretată ca $A \times (B - C)$, sau $X \div A \times B - Y$ ca $X \div (A \times (B - Y))$, iar $18-12-4$ ca $18-(12-4)$ în timp ce $^{-}19+5$ prin evaluare dă $^{-}14$, iar $-19+5$ dă $^{-}24$. Cea mai mare parte a funcțiilor scalare standard în APL sînt prezentate în tabelul 8.2.

*Tabelul 8.2*

**Funcții scalare standard în APL**

| Simbol | Nume | Exemplu de utilizare și semnificație | |
|---|---|---|---|
| | **Funcții standard diadice** | | |
| + | Adunare | X+Y | X plus Y |
| — | Scădere | X—Y | X minus Y |
| × | Înmulțire | X×Y | X înmulțit cu Y |
| ÷ | Împărțire | X÷Y | X împărțit la Y |
| * | Ridicare la putere | X*Y | X' la puterea Y |
| ⌈ | Maxim | X⌈Y | Maxim dintre X și Y |
| ⌊ | Minim | X⌊Y | Minim dintre X și Y |

*Tabelul 8.2* (continuare)

| Simbol | Nume | | Exemplu da utilizare şi semnificaţie |
|---|---|---|---|
| \| | Rest | X\|Y | X \| Y≡Y—X×⌊Y÷X<br>10 \| 17≡7 ; 1 \| 3.1416≡0.1416 ; 5 \|⁻7≡⁻3 |
| ⊗ | Logaritm | X⊗Y | Logaritm în baza X din Y |
| < | Mai mic | X<Y | Oricare din aceste funcţii ia valoarea 1 dacă relaţia este îndeplinită şi 0 în caz contrar. Pentru argumente logice (0 sau 1) X≠Y este „sau-exclusiv". |
| ≤ | Mai mic sau egal | X≤Y | |
| ≥ | Mai mare sau egal | X≥Y | |
| > | Mai mare | X>Y | |
| = | Egal | X=Y | |
| ≠ | Diferit de | X≠Y | |
| ∧ | şi | X∧Y | X∧Y≡X ⌊Y |
| ∨ | sau | X∨Y | X∨Y=X ⌈ Y |
| ⍲ | şi negat | X⍲Y | X⍲Y=~X⌊ Y |
| ⍱ | Sau negat | X⍱Y | X⍱Y=~X ⌈Y |
| ! | Combinări | X ! Y | Combinări de Y obiecte luate cîte X |

**Funcţii monadice standard**

| | | | |
|---|---|---|---|
| + | | +Y | Adună 0 cu Y (nici o schimbare) |
| — | Negare | —Y | Scade Y din 0 ; —7 =⁻7 |
| × | Signum | ×Y | Sign (Y)=0 dacă Y=0, —1 dacă Y<0 şi 1 dacă Y>0 |
| ÷ | | ÷Y | Împarte 1 la Y |
| * | | *Y | Ridică pe e la puterea Y ; e—2.71828 |
| ⌈ | Ceiling | ⌈ Y | Cel mai mic întreg ≥Y ; ⌈ ⁻3.14≡⁻3 |
| ⌊ | Floor | ⌊ Y | Cel mai mare întreg ≤Y ; ⌊ ⁻3.14≡⁻4 |
| \| | Modul | \| Y | Valoarea absolută a lui Y |
| ⊗ | Logaritm natural | ⊗Y | ln Y |
| ! | Factorial | ! Y | Y ! |
| o | | o Y | π×Y |
| ~ | Nici | ~Y | Funcţia logică NOT Y |
| ? | | ? Y | Un întreg aleator între 1 şi Y |

Se precizează că un program APL constă dintr-o combinaţie de funcţii şi argumente, pentru fiecare funcţie argumentele putînd fi în număr de unu sau două. De exemplu, în expresia X+Y, + este funcţia diadică de adunare, iar *X* şi *Y* sînt argumentele acestei funcţii.

### 8.2.3. **Construirea programelor** APL.

În cazul cel mai simplu o singură linie introdusă în sistemul APL poate fi considerată ca un program APL dacă această linie defineşte complet problema de rezolvat. Astfel de programe (formate dintr-o singură linie) sînt totdeauna introduse în sistem în *mod execuţie* (operaţiile sînt executate imediat). În cazul în care pentru a programa o problemă sînt necesare mai



Scanned with OKEN Scanner

multe linii se poate construi un program cu mai multe linii numit *procedură* sau *funcție utilizator* sau încă *funcție definită*. O funcție definită completează setul de funcții standard APL. Un astfel de program nu trebuie să fie executat imediat, ci trebuie mai întîi memorat, din care cauză el se introduce în sistem în *mod definiție*. Acest lucru se indică sistemului prin introducerea caracterului ∇ (*del*) înainte de prima instrucțiune. Sfîrșitul procedurii este marcat tot prin semnul ∇ după ultima instrucțiune.

Procedurile APL pot avea zero, unu sau doi parametri și pot sau nu să transmită o valoare (valoarea funcției). Restricția la maximum doi parametri nu este esențială deoarece un parametru poate fi chiar un vector cu mai multe componente. În funcție de numărul de parametri pot exista șase cazuri diferite de definire de proceduri (vezi tabelul 8.3.).

*Tabelul 8.3*

**Definire proceduri APL**

| Caz | Parametri apel | Valoare transmisă | Nume procedură | Instrucțiune de definire | Exemplu de utilizare |
|---|---|---|---|---|---|
| 1 | 0 | 0 | *P* | ∇*P* | *P* |
| 2 | *X* | 0 | *P* | ∇*P X* | *P X* |
| 3 | *X, Y* | 0 | *P* | ∇*X P Y* | *X P Y* |
| 4 | 0 | *Z* | *P* | ∇*Z*←*P* | *W*←*P* |
| 5 | *X* | *Z* | *P* | ∇*Z*←*P X* | *W*←*P X* |
| 6 | *X, Y* | *Z* | *P* | ∇*Z*←*X P Y* | *W*←*X P Y* |

Programul care urmează permite determinarea soluțiilor ecuației de gradul al doilea $ax^2+bx+c=0$.

```
      ∇ ECUAȚIE
[1]   DELTA←(B*2)−4×A×C
[2]    →(DELTA<0)/RIM
[3]   X1←(−B+DELTA*÷2)÷2×A
[4]   X2←(−B−DELTA*÷2)÷2×A
[5]   'VALOAREA SOLUȚIEI X1 ESTE'
[6]   X1
[7]   'VALOAREA SOLUȚIEI X2 ESTE'
[8]   X2
[9]   →0
[10]  RIM:'RĂDĂCINILE SÎNT IMAGINARE'
[11]  ∇
      A←1
      B←2
      C←¯8
      ECUAȚIE
VALOAREA SOLUȚIEI X1 ESTE
2
VALOAREA SOLUȚIEI X2 ESTE
¯4
```



Scanned with OKEN Scanner

Programul conține corpul procedurii *ECUATIE* (liniile numerotate cu 1, 2, 3, ..., 11) și un exemplu de utilizare a procedurii pentru ecuația $x^2+2x-8=0$ (următoarele patru linii). Rezultatul execuției este prezentat prin ultimele patru linii.

Instrucțiunile în cazul unei definiții de procedură sînt numerotate secvențial de sistem (în cazul prezentat de la *1* la *11*). Orice instrucțiune poate să aibă o etichetă, de fapt un identificator urmat de simbolul : (în cazul prezentat în linia *10* eticheta este *RIM*). Identificatorul ce apare ca etichetă într-o linie primește ca valoare chiar numărul liniei, putînd fi utilizat în expresii cu această valoare. De exemplu, eticheta *RIM* primește valoarea *10*. Simbolul → înseamnă GO TO, deci este un operator de salt. Astfel, o instrucțiune de forma → *RIM*, atunci cînd este executată, va avea ca rezultat transferul controlului la instrucțiunea cu eticheta *RIM*, în timp ce → *15* va transfera controlul la instrucțiunea cu numărul *15*. Instrucțiunea → *0* are ca efect transmiterea controlului în afara procedurii. Operatorul → apare întotdeauna ca prim simbol în cadrul unei linii și poate fi urmat în general de o expresie. În cazul de mai sus linia *2* descrie transferul controlului la instrucțiunea cu eticheta *RIM* dacă *DELTA* $<0$ ia valoarea 1 (adică relația este îndeplinită); în caz contrar se execută instrucțiunea *3*. Astfel, în general, o instrucțiune de forma → $(A)/B$, unde $A$ este o expresie iar $B$ o etichetă, va determina un GO TO $B$ dacă $A=1$ și va continua execuția în secvență dacă $A=0$, putîndu-se astfel realiza un salt condiționat.

8.2.4. **Tablouri în APL.** În APL tablourile pot fi cu o dimensiune (vectori), cu două dimensiuni (matrice) sau cu un număr mai mare de dimensiuni (tablouri multidimensionale).

O variabilă devine vector în APL dacă i se atribuie o listă de valori (valorile în listă fiind separate prin cîte minimum un spațiu).

De exemplu :

```
      VECTOR ← 13 17 ¯9 5
      VECTOR
13 17 ¯9 5
      V1 ← VECTOR
      V1
13 17 ¯9 5
```

Operatorii scalari se pot folosi și în cazul vectorilor. În această situație operațiile se execută componentă cu componentă.

```
      V2 ← ¯2 3 9 4
      V2+V1
11 20 0 9
      V2<V1
1 1 0 1
```



Scanned with OKEN Scanner

În cazul funcțiilor diadice, un operand poate fi un scalar, iar celălalt un vector.

```
      3*V2
 ¯6  9  21  12
      V3 ← 1 2 3 4
      2*V3
 2  4  8  16
```

Elementele unui vector pot fi referite utilizînd un indice inclus între paranteze pătrate.

```
      V2[3]
9
      V2[3]×V2[1]+V2[2]
9
```

Dacă se folosește ca indice un vector, atunci rezultatul este un nou vector obținut din elementele vechiului vector corespunzînd elementelor vectorului indice.

```
      V2[3 2 4 1 2]
9 3 4 ¯2 3
```

În cazul în care un șir de caractere este atribuit unei variabile, aceasta este tratată ca un vector cu o literă pe element.

```
      A ← 'ALFA BETA'
      A[6]
B
      A[1 2 3 4]
ALFA
```

Vectorii pot fi concatenați, virgula servind ca operator de concatenare.

```
      V ← 3 7 5 2
      V, V, 11
3 7 5 2 3 7 5 2 11
      3+4,V
7 6 10 8 5
```

Pentru sortarea crescătoare sau descrescătoare a elementelor unui vector se folosesc operatorii diadici ⍋ (*grade up*) și ⍒ (*grade down*) respectiv, care dau ca rezultat lista indicilor elementelor corespunzînd ordonării.

De exemplu:

```
      V ← 3 5 ¯7 3 2
      V1 ← ⍋V
      V1
2 5 1 3 4
```



Scanned with OKEN Scanner

```
      V[V1]
 ¯5 2 3 7 8
      V[⍋V]
 ¯5 2 3 7 8
```

Tablourile cu mai mult de o dimensiune trebuie declarate explicit. De asemenea trebuie declarate și valorile inițiale ale elementelor lor. Pentru aceasta se folosește operatorul diadic ρ (*ro*). În stînga lui ρ se dau valorile maxime ale indicilor după fiecare dimensiune sub formă de listă, iar în dreapta un scalar sau un vector reprezentînd valorile inițiale ale elementelor (pe linii).

```
      2 3 ρ 0
0 0 0
0 0 0
      M ← 2 3 ρ 11 12 13 21 22 23
      M
11 12 13
21 22 23
```

Folosit ca funcție monadică ρ dă ca rezultat un vector ale cărui elemente reprezintă dimensiunile tabloului din dreapta.

De exemplu :

```
      ρ M
2 3
```

Ca și în cazul vectorilor se pot executa operații cu tablouri, operațiile efectuîndu-se element cu element între elementele corespunzătoare. Elementele unui tablou pot fi referite prin indici plasați între paranteze și separați prin ; . Ca indici se pot utiliza și vectori.

```
      M[1 2 ; 2 3]
12 13
22 23
      M[1;]
11 12 13
      M[;2]
12
22
```

O operație foarte importantă este *reducerea*. Aceasta se scrie sub forma #/Y, unde # este o funcție standard, iar Y un tablou. De exemplu, dacă V este un vector atunci +/V înseamnă suma tuturor elementelor lui V. ⌈/V înseamnă elementul maxim din V etc.

Un vector V poate fi *comprimat* scriind W←U/V, unde U este un vector ce conține numai 1 și 0 ca elemente ; vectorul rezultat W constă din elementele lui V ce au în corespondență un 1 în U, iar dimensiunea lui W este +/U. Operația inversă de *extensie* este notată cu V←U\W, unde U este un vector ce are ca elemente doar 1 și 0, numărul de elemente 1 fiind



Scanned with OKEN Scanner

egal cu numărul de elemente ale vectorului $W$. Vectorul rezultat $V$, a cărui lungime este egală cu a lui $U$, are elementele lui $W$ în poziţiile ce corespund la 1 în $U$ şi zero unde $U$ are zero. Dacă $V$ este o matrice, atunci $U/V$ comprimă fiecare linie a lui $V$ funcţie de vectorul $U$, iar $U/[1]\ V$ comprimă fiecare coloană a lui $V$ funcţie de $U$. Analog pentru extensie.

*Produsul intern* între două matrice este notat cu $+.\times$ ; de exemplu, $C \leftarrow A +.\times B$ înseamnă :

$$C[I\,;\,J] \equiv +/A[I\,;] \times B\ [;\,J]$$

*Produsul extern* între tablouri, notat cu $A \circ .\# B$, unde # este o funcţie standard, cere ca operaţia # să fie aplicată la fiecare element al lui $A$ cu fiecare element al lui $B$, generînd astfel un tablou de dimensiune mai mare.

De exemplu :

```
      1 3 5 o.x 2 4 6
 2  4  6
 6 12 18
10 20 30
```

Operaţiile cu tablouri prezentate mai înainte nu epuizează întreaga gamă de operaţii pe care o pune la dispoziţie limbajul APL. Totuşi se consideră că acestea sînt suficiente pentru a pune în evidenţă puterea şi eleganţa limbajului APL.

## 8.3. LIMBAJUL TTPL

Limbajul TTPL [5, 114] este o versiune a limbajului APL implementată pe calculatorul FELIX C-256, apelînd la terminale conversaţionale uzuale. Ca urmare, operatorii APL au fost adaptaţi la semnele specifice claviaturii terminalelor existente. Limbajul APL fiind prezentat anterior, în cele ce urmează se vor prezenta doar diferenţele dintre TTPL şi APL.

1. Operatorii de atribuire şi de salt se reprezintă prin semnul $\leftarrow$. Diferenţierea dintre ei se face ţinînd seama de faptul că operatorul de salt apare pe prima poziţie într-o instrucţiune, în timp ce operatorul de atribuire are obligatoriu în stînga un operand.

2. Nu există simboluri distincte pentru semnul minus şi operatorul de scădere.

3. În şirurile de caractere apostroful este admis dacă este dublat.

4. Funcţiile scalare standard în TTPL sînt prezentate în tabelul 8.4.

5. Pentru marcarea situaţiilor în care sistemul TTPL este în mod definire se foloseşte simbolul %. Cu acesta deci începe şi se termină o funcţie definită sau o procedură TTPL.



Scanned with OKEN Scanner

*Tabelul 8.4*

**Funcții scalare standard în TTPL**

| Simbol | Nume | Exemplu de utilizare și semnificație | |
|---|---|---|---|
| | **Funcții standard diadice** | | |
| + | Adunare | X+Y | X plus Y |
| − | Scădere | X−Y | X minus Y |
| * | Înmulțire | X*Y | X înmulțit cu Y |
| : | Împărțire | X : Y | X împărțit cu Y |
| ↑ | Ridicare la putere | X ↑ Y | X la puterea Y |
| = | Egal | X=Y | Aceste funcții iau valoarea 1 dacă relația este îndeplinită și 0 în caz contrar |
| ≠ | Diferit de | X≠Y | |
| < | Mai mic | X<Y | |
| =< | Mai mic sau egal | X=<Y | |
| > | Mai mare | X>Y | |
| =< | Mai mare sau egal | X=>Y | |
| ++ | Sau | X++Y | X și Y operanzi logici |
| ** | Și | X**Y | |
| | **Funcții standard monadice** | | |
| + | Modul | +Y | Valoarea absolută a lui Y |
| − | Nici sau minus | −Y | Scade Y din 0 dacă Y este număr și negare Y dacă Y este operand logic |
| * | Signum | *Y | sign (Y) =0 dacă Y=0, −1 dacă Y<0, 1 dacă Y>0 |
| ! | Inversare | !Y | Împarte 1 la Y |
| ↑ | Exponențiere | ↑Y | $e^Y$ ; e=2.71828 |

6. Etichetele de instrucțiuni în proceduri TTPL sînt succesiuni de litere și cifre incluse între paranteze pătrate. Salturile condiționate se pot programa utilizînd instrucțiuni de forma ←A * E, unde A este un număr de linie sau o etichetă, iar E o expresie. De exemplu, în cazul instrucțiunii ←5*I<=N se va executa salt la instrucțiunea cu numărul 5 atît timp cît funcția I<=N ia valoarea 1 (adică relația este îndeplinită); în caz contrar se va executa instrucțiunea ce urmează imediat în secvență.

7. Ca și funcțiile definite din APL, o funcție definită în TTPL poate fi recursivă; deci poate folosi în corpul ei propria definiție.

În continuare se dau două exemple de funcții definite recursiv: prima funcție calculează factorial de $N$ după relația $N! = N.(N-1)!$, iar a doua calculează combinări de $N$ obiecte luate cite $k$ după relația $C_N^k = C_{N-1}^k + C_{N-1}^{k-1}$.

```
      % Z ← FACTORIAL N
[1]   Z ← 1
[2]   ← O* (N=0)
```

```
      % CNK ← N COMB K
[1]   CNK ← 1
[2]   ← O* ((K=N)++(K=0))
```

```
[3]    Z ← N*FACTORIAL N-1
[4]    %
       FACTORIAL 5
120
```

```
[3]  CNK ← ((N-1) COMB (K-1))+((N-1)
     COMB K)
[4]  %
     5 COMB 3
10
```

8. Un *tablou vid* în TTPL este un tablou cu toate dimensiunile nule.

9. Pentru determinarea dimensiunilor unui tablou se folosește funcția monadică $.

```
      X ← 11 1 -7
      $X
 3
      Y ← 'ALFA'
      $Y
 4
```

Funcția diadică $ se folosește pentru declararea și inițializarea de tablouri.

```
      M ← 3 3 $ 1 0 0 0
      M
1 0 0
0 1 0
0 0 1
```

10. Utilizarea indicilor se face la fel ca în APL.

11. Funcțiile standard scalare se extind la tablouri în următoarele moduri :

- element cu element ;
- reducere (operație notată cu /) ;
- produs intern care între matrice se notează cu $+.*$ și care se poate extinde pentru oricare două funcții diadice $F$ și $G$ sub forma $F.G$ ;
- produs extern notat cu &.F, unde F este o funcție standard.

Sînt de asemenea definite operațiile de comprimare și extensie (sau dilatare) ca și în APL, cu observația că operatorul pentru extensie s-a notat cu //.

Ca operator diadic, virgula înseamnă concatenare, în timp ce folosită ca funcție monadică dă operația de desfășurare (*ravel*).

```
      M ← 3 3 $ 1 0 0 0
      ,M
1 0 0 0 1 0 0 0 1
```

Scanned with OKEN Scanner

# 9

# LIMBAJE SPECIALIZATE

Limbajele specializate sînt destinate rezolvării problemelor specifice unor domenii de activitate şi, de obicei, au la bază o mulţime de funcţii prin referirea cărora se rezolvă problemele din domeniul pentru care au fost concepute. Acestea sînt limbaje de tip neprocedural.

## 9.1. LIMBAJE DE SIMULARE

Simularea este tehnică constînd în realizarea unei reprezentări (model) pentru fenomene reale, cu scopul studierii modelului în locul fenomenului. Complexitatea modelelor a condus la utilizarea pe scară largă a calculatoarelor electronice. Cum programarea în limbajele universale a devenit o problemă, s-a trecut la realizarea de limbaje destinate special simulării.

O clasă importantă de limbaje specializate o constituie limbajele de simulare care se împart în două categorii : cele care privesc procesele continue (limbaje specifice calculatoarelor analogice), cele care privesc procesele discrete (limbaje specifice calculatoarelor numerice). Limbajele din categoria a doua au o arie de răspîndire mai mare faţă de cele din prima categorie. Aceasta pentru motivul că simularea „continuă" necesită cunoştinţe de strictă specialitate din partea utilizatorului.

Unul dintre limbajele din prima categorie, implementate pe sisteme IBM încă din 1958, este DYANA (DYnamics ANAlizer) şi are la bază limbajul FORTRAN, fiind destinat descrierii sistemelor dinamice. Acestuia i-au urmat altele : DYSAC (DigitallY Simulated Analog Computer), DAS (Digital Analog Computer) etc. [74].

La rîndul lor, limbajele pentru simulare discretă sînt de două tipuri : cele care au la bază un număr de blocuri (cu funcţiuni precise) prin asamblarea cărora rezultă sistemul ce se simulează şi cele care au la bază un set de instrucţiuni.



Scanned with OKEN Scanner

Din limbajele destinate simulării discrete, GPSS (General Purpose Simulation System) este cel mai răspîndit. De fapt, GPSS constituie un sistem general de simulare, utilizat curent la calculatoare IBM. Modelele GPSS apar ca succesiuni de blocuri (GPSS dispun de peste 40 de tipuri de blocuri), funcțiunile blocurilor fiind îndeplinite de subprograme corespunzătoare. Există și posibilitatea utilizării de subprograme externe sistemului GPSS, grație blocului HELP.

Pentru calculatoarele din gama FELIX s-a creat limbajul SIMUB (limbaj de SIMulare al Universității din București) realizat după modelul lui GPSS [107].

Este știut că atunci cînd se realizează modelul unui sistem, unele variabile sînt cunoscute, iar altele nu. Scopul care se urmărește este acela de a exprima (folosind interacțiunile dintre componentele sistemului) ceea ce nu se cunoaște în funcție de ceea ce se cunoaște. Cum, uneori, acest lucru nu este posibil să se exprime prin relații matematice, se recurge la relații logice. Modelul de simulare discretă (luînd în considerație și aceste condiții), pornind de la valori ale parametrilor și variabilelor cunoscute, cu un algoritm de simulare, produce valori pentru ceea ce nu se cunoaște. Modificările variabilelor rezultă datorită apariției unor evenimente în sistem. Așadar, un algoritm de simulare ține seama de componentele sistemului ce se simulează, de variabilele logice care descriu stările componentelor și de o „agendă" (memorarea evenimentelor ce se produc și rutina de generare a evenimentelor). Evidența ordinii de apariție a evenimentelor în sistem este asigurată de o variabilă numită *ceasul simulării*. De fiecare dată cînd ceasul crește, se analizează agenda evenimentelor și stările sistemului, ceea ce duce la decizii în funcție de noile evenimente (produse, anulate, modificate), cu scopul modificării stărilor componentelor sistemului (ceasul poate crește în două moduri : constant sau aleator). Cum orice algoritm de simulare duce la producerea unor valori statistice referitoare la ieșiri, algoritmul de simulare trebuie să aibă în vedere astfel de prelucrări.

Simularea se aplică la studiul unor sisteme complexe din cele mai variate domenii (procese de producție în fabrici, trafic urban, aerian și naval, telecomunicații, rețele de calculatoare etc.). Limbajele de simulare sînt create cu scopul de a ușura atît modul de gîndire în realizarea de modele de simulare discretă cît și programarea pe calculator. La majoritatea limbajelor de simulare se face distincție între entitățile pasive (de exemplu piesele ce se realizează) și cele active (mașinile cu care se realizează piesele). Entitățile pasive se numesc *tranzacții*. Limbajul de simulare este acela care permite realizarea unui model în vederea introducerii tranzacțiilor pentru a le studia comportarea. Entitățile active sînt cele care asigură circulația tranzacțiilor.



Scanned with OKEN Scanner

Limbajul SIMUB dispune de 14 entităţi, cărora li se asociază atribute standard (numerice sau logice). În fapt, acestea sînt variabile prin care se caracterizează stările sau componentele sistemului. Prin simulare, entităţile interacţionează şi conduc la modificări ale valorilor atributelor. *Entităţile* limbajului SIMUB sînt de următoarele tipuri : de bază (blocuri, tranzacţii), de echipament (memorii, comutatori), de calcul (variabile aritmetice, booleene, funcţii), statistice (fire de aşteptare, tablouri de frecvenţă), de referinţă (celule de păstrare, matrice de păstrare), de tip lanţ (lanţul evenimentelor curente, viitoare, al utilizatorului, al tranzacţiilor întrerupte, al celor în stare de aşteptare, lanţ de întîrziere). *Blocurile* sînt cele mai importante entităţi. Cu ele se descriu modificările ce pot apărea în sistem. Pot fi blocuri de mişcare a tranzacţiilor prin facilităţi, de generare a tranzacţiilor, de distrugere de tranzacţii, de control logic, de modificare a parametrilor, de calcul al unor statistici, de listare. Descrierea unui bloc se face cu o cartelă care indică numele blocului şi o listă de parametri. O tranzacţie ce a pătruns într-un bloc (parcugerea blocurilor este secvenţială, exceptînd blocurile care dau salt) este supusă prelucrării prin subprogramul asociat blocului.

Descrierea entităţilor limbajului SIMUB se face, de obicei, printr-o singură cartelă. Rutinele sînt scrise în ASSIRIS [107].

O serie de facilităţi nu s-au implementat încă la această versiune a limbajului. De exemplu, urmează să se implementeze o serie de entităţi statistice care să permită o analiză bidimensională (tabele de contingenţă, tabele de corelaţie).

Cu ocazia elaborării limbajului, s-a creat şi o bibliotecă de subrutine FORTRAN pentru generarea unor tipuri de variabile aleatoare (biblioteca RAVAGE), cu scopul de a mări acurateţea de generare a tranzacţiilor [90].

Limbajul SIMUB se poate implementa pe orice sistem de calcul FELIX C-256. În cazul în care modelul nu necesită dimensiuni mari pentru tablouri, fire de aşteptare, celule şi matrice de cuvinte păstrate, implementarea este posibilă cu o memorie de 64 K. Aplicaţii de dimensiuni mai mari se pot efectua numai pe sisteme cu memorie de peste 128 K.

Versiunea SIMUB actuală pune la dispoziţia utilizatorului următoarele tipuri de cartele : cartelele pentru controlul simulării : SIMULATE, JOB, END, START, CLEAR, RESET ; cartela de realocare a entităţii limbajului SIMUB (REALLOCATE) ; cartele de definire şi dimensionare a entităţilor de tip tablouri (QTABLE, TABLE), matrice de cuvinte păstrate (MATRIX), depozite (STORAGE) ; cartele de iniţializare a comutatorilor logici, a cuvintelor de păstrare şi a matricelor de cuvinte păstrate (INITIAL), a generatorilor de numere aleatoare (RMULT) ;

cartele de definite a entităților de calcul (FUNCTION, VARIABLE, BVARIABLE); cartele de definire a blocurilor (ADVANCE, ASSIGN, ALTER, ASSEMBLE, BUFFER, CHANGE, COUNT, DEPART, ENTER, EXAMINE, EXECUTE, GATE, GATHER, GENERATE, INDEX, JOIN, LEAVE, LINC, LOGIK, LOOP, MARK, MATCH, MASAVEVALUE, PREEMPT, PRINT, PRIORITY, QUEUE, RELEASE, REMOVE, RETURN, SAVEVALUE, SCAN, SEIZE, SELECT, SPLIT, TABULATE, TEST, TERMINATE, TRANSFER, TRACE, UNLINK, UNTRACE). De asemenea este inclusă facilitatea de referire simbolică a entităților de tip bloc.

Pentru descrierea detaliată a sintaxei și acțiunii acestor cartele, precum și pentru unele exemple se recomandă „Manualul de aplicare" [107].

## 9.2. LIMBAJE DE INTEROGARE

Sfera noțiunii de *limbaj de interogare* este foarte cuprinzătoare întrucît se întrepătrunde cu noțiunile de limbaj de regăsire a informației, sistem de gestiune a bazelor de date, limbaj natural și limbaj conversațional. În consecință, diversitatea acestor limbaje este considerabilă, la o extremă aflîndu-se limbajele cu organizare a informației în format fix și modalități reduse și rigide de acces la informație, iar la cealaltă extremă, limbajele conversaționale bazate pe texte într-un limbaj natural (în general, engleza).

Adesea, se asociază două limbaje la același sistem : unul pentru gestiunea bazei de date și altul pentru regăsirea informației, acesta din urmă fiind mai apropiat de limbajul natural, ceea ce impune ca limbajul să trateze trei aspecte : *sintaxa* (pentru evidențierea structurii gramaticale a întrebării), *semantica* (pentru stabilirea sensului cel mai probabil) și *regăsirea* propriu-zisă (aspect strîns legat de organizarea datelor).

Diversitate mare se întîlnește și în privința domeniilor de utilizare (aplicații militare, rezultate sportive, informații din enciclopedii etc.).

Un limbaj cu exploatare on-line este ADAM (A generalized Data Management System), operațional din 1965. Ceea ce are nou acesta față de altele este faptul că dispune de un translator sintactic discret, fapt ce permite utilizatorului inițiativă : definirea de noi limbaje, restricții de limbaje, extensii de limbaje.

Scanned with OKEN Scanner

# 10

# UTILIZAREA COMBINATĂ A LIMBAJELOR ASSIRIS, COBOL ȘI FORTRAN

În cadrul acestui capitol vor fi prezentate succint unele aspecte privind segmentarea programelor, cu particularizări pentru programele ASSIRIS, COBOL și FORTRAN, precum și utilizarea combinată, în scrierea programelor, a subprogramelor ASSIRIS, COBOL și FORTRAN.

## 10.1. SEGMENTAREA PROGRAMELOR

Un program pentru a fi executat trebuie încărcat (înregistrat) în memoria centrală. Uneori însă, dimensiunea zonei-memorie necesară încărcării programului este insuficientă. În astfel de situații, pentru a fi executat, programul poate fi structurat (divizat) în unități de program distincte, numite *segmente*, dintre care unele vor fi încărcate și executate în aceeași zonă de memorie. Acest lucru este posibil deoarece, în diverse momente ale execuției programului, este necesară prezența în memorie numai a anumitor segmente (vezi 2.5.9).

Utilizarea acestei tehnici de programare, numită segmentare, necesită ca, încă din faza de elaborare a programelor, acestea trebuie în așa fel structurate, încît componentele să realizeze funcții bine precizate.

În format sursă, componentele programului reprezintă, în general, *unități independente*. Una din aceste unități de program este un *program principal*, iar celelalte, *subprograme*. În raport cu subprogramele, programul principal are rolul central, asigurînd, în principiu, apelul subprogramelor pentru execuție, astfel încît să fie respectată logica programului

În format executabil, componentele programului reprezintă *segmente* Segmentul ce corespunde programului principal se numește segment rădăcină ; el este încărcat primul în memoria centrală și rămîne în memorie pe toată durata execuției programului. Segmentele ce corespund subprogramelor sînt încărcate în memoria centrală ulterior, în zone de



Scanned with OKEN Scanner

memorie distincte sau comune ; momentele în care sînt încărcate aceste segmente corespund momentelor apelării pentru execuţie a subprogramelor.

10.1.1. **Construirea segmentelor. Tipuri de segmente.** Segmentele sînt constituite, în general, în faza de editare a legăturilor din textele modulelor obiect, obţinute prin traducerea programelor sursă. Indiferent de limbajul în care este scris programul sursă, în faza de compilare (la sistemul FELIX C-256) pot fi generate trei tipuri de module : *module de program*, *module de date* şi *module de date comune*. Dimensiunea maximă a unui modul este 64 K baiţi (1bait=8 biţi). Tipul modulelor, conţinutul acestora, numele care le sînt atribuite sînt funcţie de limbajul în care este scris programul. În general, modulele de program conţin instrucţiuni în limbaj-maşină, modulele de date, date utile unui modul de program, iar modulele de date comune, date utile mai multor module de program.

Segmentele sînt de trei tipuri, corespunzătoare tipului de module pe care le conţin : *segmente de program* care grupează unul sau mai multe module de program, *segmente de date* care grupează un singur modul de date şi *segmente de date comune* care grupează unul sau mai multe module de date comune (gruparea mai multor module de date comune trebuie indicată în cartela TREE). Dimensiunea maximă a unui segment este 64 K baiţi.

Structura segmentelor poate fi stabilită de utilizator, prin intermediul cartelelor SEG, sau, dacă acestea nu sînt precizate, de editorul de legături ; modulele care alcătuiesc un segment pot proveni din compilări (asamblări) realizate în aceeaşi lucrare în care se realizează şi editarea legăturilor, din biblioteci de tip BT, din fişierul de intrare al sistemului (cazul modulelor obiect perforate pe cartele).

10.1.2. **Cartela** TREE. Structura unui program este indicată editorului de legături prin intermediul cartelei TREE, al cărei format este :

.TREE *expresie*

În *expresie* pot apărea nume de segmente şi simbolurile (, ), şi +. Numele segmentelor sînt formate din maximum opt caractere, primul fiind o literă.

Semnificaţia simbolurilor este următoarea :

– ( segmentul citat la dreapta acestui simbol este de nivel inferior celui citat la stînga ;

– ) segmentul citat la dreapta acestui simbol este de nivel superior celui citat la stînga ;

– , segmentele citate la dreapta şi la stînga acestui simbol sînt de acelaşi nivel ;



Scanned with OKEN Scanner

– + segmentul citat la dreapta acestui simbol este un segment de date comune.

Numai segmentele de program și de date comune pot fi citate în cartela TREE, iar primul segment este întotdeauna segmentul rădăcină. În absența cartelei TREE, editorul de legături înlănțuie segmentele liniar.

**Exemplu.** Pentru structura arborescentă din fig. 10.1, unde *SP1*, *SP2*, *SP3*, *SP4* și *SP5* sînt numele unor segmente de program, *SD1* și *SD2* sînt numele unor segmente de date, iar *SC1* și *SC2* sînt numele unor segmente de date, comune, cartela TREE trebuie scrisă astfel :

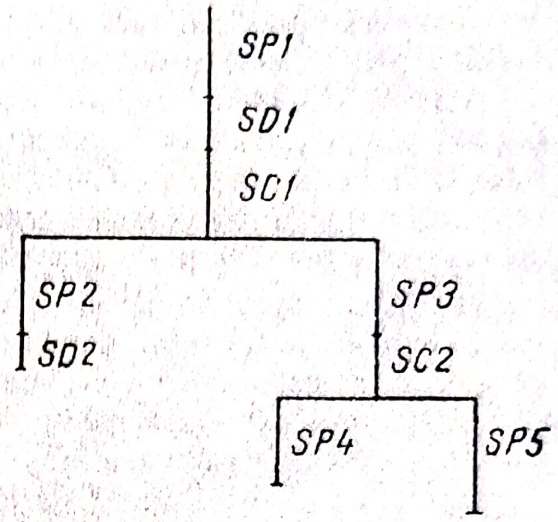



Fig. 10.1. Exemplu de program segmentat.

TREE SP1+SC1 (SP2, SP3↓SC2 (SP4, SP5))

10.1.3. **Segmentarea programelor** FORTRAN. Prin compilarea unei unități de program (program principal sau subprogram) se obține : un modul de program și unul sau mai multe module de date comune (egal cu numărul de blocuri de tip „common" definite în program).

*Modulul de program* conține instrucțiunile în limbaj-mașină ce corespund instrucțiunilor FORTRAN, variabilele locale și constantele definite în unitatea de program respectivă, precum și date incluse implicit de compilator, care reprezintă constante și tabele necesare execuției programului, rezultate intermediare ale unor operații etc. ; numele atribuit modulului de program este F%MDATA, dacă unitatea de program este un program principal, sau simbolul citat în instrucțiunea SUBROUTINE, dacă unitatea de program este un subprogram. *Modulul de date comune* conține variabilele grupate într-un bloc de tip „common", etichetat sau nu ; numele atribuit modulului de date comune este F%BLK, dacă blocul care îi corespunde nu este etichetat, sau eticheta asociată acestui bloc, în caz contrar.

În faza de editare a legăturilor, dacă utilizatorul nu specifică, prin cartela SEG, gruparea modulelor în segmente, editorul de legături grupează modulele obținute (care pot fi module de program și module de date comune) într-un singur segment de program, căruia îi atribuie numele F%MDATA.

În legătură cu dimensiunea programului executabil obținut, pot apărea două situații :

– aceasta poate fi mai mare decît dimensiunea maximă a unui segment (64 K baiți) ;

– aceasta poate fi mai mare decît cea a zonei de memorie (partiția) alocată pentru execuția programului.

Ambele situații, care sînt semnalate de editorul de legături și, respectiv, încărcător, implică segmentarea programului. Tehnica ce trebuie folosită în acest scop este funcție de compoziția programului (de ponderea instrucțiunilor, a variabilelor locale sau a celor grupate în blocuri de tip „common"), precum și de gradul său de modularizare.

a) Astfel, dacă ponderea instrucțiunilor în cadrul programului este mare și dacă programul nu conține variabile grupate în blocuri de tip „common" și este structurat, modulele de program obținute la compilare pot fi grupate prin intermediul cartelei SEG în mai multe segmente de program, care prin intermediul cartelei TREE pot fi înlănțuite arborescent. În cazul în care programul nu este structurat, trebuie separate funcțiile acestuia și rezolvate în cadrul unor unități de program distincte. Trebuie însă avut în vedere ca înlănțuirea segmentelor să se realizeze în așa fel, încît segmentele situate pe ramuri diferite să nu se apeleze între ele.

b) Dacă în program există variabile grupate în blocuri de tip „common", iar dimensiunea acestora este relativ mare, modulele de date comune, ce corespund acestor blocuri, pot fi grupate în cadrul unor segmente de date comune distincte (unul pînă la patru segmente). Această grupare trebuie indicată compilatorului FORTRAN prin intermediul directivei SEGMENT, al cărei format este

*SEGMENT G1,G2, . . . ,G*n*

G*i* ($1 \leqslant i \leqslant 4$) reprezintă grupări de module de date comune (de forma : C1+C2+ . . . +C*p*, unde C*j* ($1 \leqslant j \leqslant p$) reprezintă numele acestora) care vor constitui, fiecare, un segment de date comune. Numele atribuit acestui segment reprezintă numele primului modul al grupării.

Directiva SEGMENT, care urmează imediat cartelei COMPILE, trebuie să fie specificată pentru fiecare unitate de program în care există grupări de variabile în blocuri de tip „common". Grupările de module indicate în directivele SEGMENT asociate unui program, trebuie să fie respectate pentru toate unitățile de program în care sînt definite grupări ce specifică căror segmente de program le sînt asociate segmentele de date comune (unui segment de program îi pot fi asociate maximum patru segmente de date comune).

Dacă în directiva SEGMENT nu este specificată nici o grupare, atunci toate modulele de date comune vor fi grupate într-un singur segment, care va primi numele primului modul de date comune, generat prin traducerea unității de program respective.

c) Dacă într-o unitate de program nu sînt definite blocuri de tip „common", iar ponderea variabilelor locale este relativ mare, atunci



Scanned with OKEN Scanner

variabilele locale, precum și datele incluse implicit de compilator pot fi grupate într-un modul de date distinct. Această grupare este indicată compilatorului prin directiva SEGDATA care are două formate :

*SEGDATA – utilizat numai într-un program principal ;

*SEGDATA (nume) – utilizat numai într-un subprogram.

Numele atribuit de compilator modulului de date generat este F% MDATA, în cazul unui program principal, sau cel indicat în directiva SEGDATA, în cazul unui subprogram. De asemenea, modulul de program (ce conține numai instrucțiuni în limbaj-mașină), căruia îi este asociat modulul de date, i se atribuie numele F%MAIN, în cazul unui program principal, și numele subprogramului, în cazul unui subprogram. În faza de editare a legăturilor, din fiecare tip de modul (de program, de date) este generat un segment de același tip. Segmentul de date, fiind asociat segmentului de program, va primi numele acestuia (nume care trebuie indicat printr-o cartelă SEG).

Dacă pentru o unitate de program este necesară atît generarea de segmente de date cît și de segmente de date comune, corespunzător tipului unității de program, poate fi utilizat unul dintre următoarele formate ale directivei SEGDATA :

*SEGDATA G1, G2, ... ,G*n* – în cazul unui program principal ;

*SEGDATA (nume) G1, G2, ... ,G*n* – în cazul unui subprogram.

G*i* ($1 \leqslant i \leqslant 3$) au aceeași semnificație ca în cazul directivei SEGMENT.

10.1.4. **Segmentarea programelor** COBOL. Două metode pot fi utilizate pentru segmentarea programelor COBOL.

a) Prima metodă, care permite obținerea unei structuri arborescente cu mai multe nivele, necesită structurarea programului în unități de program distincte (program principal, subprograme) și indicarea modului de înlănțuire (ierarhia) a segmentelor ce corespund acestor unități de program.

Alcătuirea și compoziția segmentelor în cazul programelor COBOL este oarecum particulară, deoarece acestea nu sînt precizate prin cartele SEG, ci sînt stabilite de compilator.

Astfel, prin compilarea unei unități de program rezultă un modul de program, un modul de date și unul sau mai multe module de date comune (egal cu numărul fișierelor care nu utilizează unitățile standard ale sistemului).

*Modulul de program* conține instrucțiunile în limbaj-mașină ce corespund instrucțiunilor COBOL. *Modulul de date* conține datele definite

în diviziunea DATA precum și datele incluse implicit de compilator în programul obiect (constantele necesare execuției programului, tabelele de descriere a fișierelor, literalele specificate în diviziunea PROCEDURE, rezultatele intermediare ale unor operații etc). *Modulul de date comune* conține zonele tampon asociate unui anumit fișier.

Numele modulelor generate sînt stabilite prin adăugarea la primele șase caractere ale numelui programului (indicat în paragraful PROGRAM-ID) a :

– caracterelor 01, în cazul modulului de program ;
– caracterelor 00, în cazul modulului de date ;
– identificatorul de exploatare al fișierului (o literă urmată de una din cifrele 1, 2, 3 sau 4, care indică tipul prelucrării acestuia), în cazul modulului de date comune (dacă numele programului este format din mai puțin de șase caractere, se completează pînă la șase cu caracterul %).

Compilatorul COBOL asociază modulelor generate o cartelă SEG, care indică numele ce va fi atribuit, în faza de editare a legăturilor, segmentelor de program și de date ; acest nume coincide cu numele modulului de date generat.

Din textele modulelor obiect rezultate, editorul de legături va genera : un segment de program, care conține modulul de program, subprograme din BSS (Biblioteca de Subprograme Standard), module de acces SGF etc., ; un segment de date, care conține modulul de date ; cîte un segment de date comune din fiecare modul de date comune. Segmentul de program și segmentul de date primesc numele specificat în cartela SEG, iar segmentele de date comune, numele modulelor de date comune pe care le conțin.

Modul de înlănțuire a segmentelor programului trebuie precizat de utilizator prin cartela TREE, în care vor fi citate numele segmentelor de program.

*b)* A doua metodă, care permite obținerea unei structuri arborescente cu două nivele, necesită gruparea instrucțiunilor în secțiuni distincte (care să realizeze funcții distincte ale programului). Între secțiuni trebuie să existe o ierarhie, care este pusă în evidență prin intermediul unor *numere de prioritate*, ce sînt asociate secțiunilor. Numerele de prioritate pot lua valori în mulțimea {0, 1, . . . ,99} și sînt indicate în titlul secțiunilor, conform formatului :

*nume-secțiune* SECTION *număr-prioritate.*

Pot exista mai multe secțiuni cu același număr de prioritate, iar dacă unor secțiuni nu le-au fost atribuite astfel de numere, compilatorul le atribuie prioritatea 0. Pe baza numerelor de priorietate, compilatorul va iniția segmentarea programului. Prin compilarea unui program astfel secționat sînt generate : unul sau mai multe module de program

(care este funcție de numerele de prioritate asociate secțiunilor), un modul de date și un număr de module de date comune (care este funcție de numărul fișierelor definite în program). Conținutul și numele atribuite modulelor de date și de date comune sînt stabilite după aceleași reguli ca și în cazul modulelor de același tip, obținute prin compilarea unui program nesecționat în aceat fel.

Un modul de program poate conține : instrucțiunile în limbaj-mașină ce corespund instrucțiunilor din secțiunile cu numere de prioritate mai mici decît 50, caz în care numele modulului este stabilit adăugînd 01 la primele șase caractere ale numelui programului, sau instrucțiunile în limbaj-mașină ce corespund instrucțiunilor din una sau mai multe secțiuni ce au același număr de prioritate, mai mare decît 49, caz în care numele modulului este stabilit prin adăugarea la primele cinci caractere ale numelui programului a caracterului % și a unui număr format din două cifre, care reprezintă diferența dintre numărul de prioritate al secțiunii (secțiunilor) și 48. Totodată, compilatorul asociază fiecărui modul de program o cartelă SEG, care indică numele ce va fi atribuit segmentelor de program și de date generate.

Editorul de legături generează din fiecare modul un segment ce corespunde tipului modulului. Segmentul de program, care conține modulul de program ce corespunde secțiunilor cu numere de prioritate mai mici decît 50, reprezintă *segmentul rădăcină* și, împreună cu segmentul de date, vor primi ca nume, numele modulului de date, iar segmentele de date comune primesc numele modulelor de date comune pe care le conțin. Numele segmentelor de program care conțin modulele de program ce corespund secțiunilor cu numere de prioritate mai mare decît 49 sînt stabilite astfel : la primele cinci caractere ale numelui programului se adaugă caracterul % și un număr format din două cifre, care reprezintă diferența dintre numărul de prioritate al secțiunilor și 49.

Segmentele generate sînt dispuse pe două nivele. La primul nivel sînt situate segmentul de program ce corespunde secțiunilor cu numere de prioritate mai mici decît 50, segmentul de date și segmentele de date comune ; aceste segmente alcătuiesc rădăcina programului. La al doilea nivel sînt situate celelalte segmente de program, numite *segmente independente*, care vor fi încărcate și executate în aceeași zonă de memorie.

Segmentele independente au acces la segmentul de date și la segmentele de date comune. De asemenea, segmentele independente se pot apela reciproc și pot apela execuția unor secvențe de instrucțiuni din segmentul rădăcină.

La nivelul programului sursă, comunicarea între secțiuni se realizează prin instrucțiuni GO TO sau PERFORM (în care pot fi citate nume de paragrafe sau de secțiuni).



Scanned with OKEN Scanner

Deoarece segmentele independente nu pot coexista în memorie, utilizarea instrucțiunii PERFORM impune respectarea următoarelor reguli :

– o instrucțiune PERFORM, aflată într-o secțiune cu număr de prioritate $p$ ($p>49$) nu poate apela paragrafe din secțiuni cu numere de prioritate mai mari decît $p$ (în schimb, poate apela paragrafe din secțiuni cu numere de prioritate egale cu $p$ sau mai mici decît 50) ;

– o instrucțiune PERFORM aflată într-o secțiune cu număr de prioritate mai mic decît 50, poate apela paragrafe conținute într-o singură secțiune, cu număr de prioritate mai mare decît 49.

Pentru optimizarea programului, este bine ca secțiunile cu numere de prioritate mai mari decît 49 să grupeze instrucțiunile executate mai puțin frecvent ; în acest fel se micșorează timpul necesar încărcării în memorie a segmentelor independente ce corespund acestor secțiuni.

10.1.5. **Segmentarea programelor** ASSIRIS. În cazul programelor ASSIRIS, segmentarea implică secționarea textelor sursă, care se realizează prin intermediul directivei CSECT. În funcție de argumentul citat în această directivă (P, D sau C) pot fi definite trei tipuri de secțiuni : de program, de date sau de date comune ; în cadrul textului sursă secțiunile de același tip trebuie să fie grupate.

În urma prelucrării unui text sursă secționat, asamblorul generează următoarele tipuri de module obiect :

– *un modul de program*, din toate secțiunile de program ;

– *un modul de date*, din toate secțiunile de date ;

– *cîte un modul de date comune* din fiecare secțiune de date comune.

Conținutul acestor module este stabilit, în general, de utilizator ; în principiu, *modulul de program* conține instrucțiuni în limbaj-mașină, *modulul de date*, date utile (constante și variabile) modulului de program, iar *modulul de date comune*, date necesare mai multor module de program.

Numele modulelor pot fi stabilite de utilizator, în cazul în care directivele CSECT sînt etichetate (dacă mai multe secțiuni de program sau de date concură la obținerea unui modul, numele acestuia va fi eticheta directivei CSCECT a primei secțiuni întîlnită de asamblor), sau de asamblor, în caz contrar.

Segmentele generate în faza de editare a legăturilor corespund tipurilor modulelor. În cadrul unui segment pot fi grupate unul sau mai multe module de același tip. Compoziția segmentelor trebuie indicată prin cartele SEG, care precizează totodată și numele ce vor fi atribuite segmentelor de program și segmentelor de date ce le sînt asociate. În cazul segmentelor de datę comune, numele acestora corespund numelor modulelor pe care le conțin (dacă într-un segment de date comune sînt grupate mai multe module, segmentul primește numele primului modul, iar gruparea trebuie indicată în cartela TREE).



Scanned with OKEN Scanner

Realizarea de programe cu structuri arborescente, necesită utilizarea cartelei TREE, în care trebuie citate numai segmentele de program și de date comune.

Pentru fiecare program executabil generat, editorul de legături alcătuiește o tabelă, numită *tabela segmentelor*, care conține adresele zonelor în care urmează să fie încărcate, pentru execuție, segmentele programului (vezi 11.3.3). Primul cuvînt al tabelei segmentelor reprezintă adresa zonei de memorie în care sînt înregistrate informații generale asupra programului (numele, data, tipul etc.), iar următoarele cuvinte, adresele zonelor de încărcare a segmentelor. De asemenea, fiecărui segment de program ce este apelat pentru execuție de către un alt segment de program, îi este asociată o tabelă, numită *tabela punctelor de intrare*, care conține adresele instrucțiunilor de la care va începe execuția segmentului apelat. Definirea punctelor de intrare într-un segment de program se realizează prin cartela ENTSG.

La încărcarea segmentului rădăcină în memorie, se mai realizează următoarele operații :

– încărcarea tabelei segmentelor ;

– încărcarea în registrul R14 a adresei de început (de bază) a segmentului rădăcină ;

– încărcarea în registrul R15 a adresei tabelei segmentelor.

10.1.5.1. *Comunicarea între secțiuni.* Tipurile de comunicări ce pot fi realizate între secțiunile unui program sînt :

– într-o secțiune de program sînt utilizate date definite într-o altă secțiune de program, de date sau de date comune ;

– într-o secțiune de program se transferă controlul unor instrucțiuni aflate într-o altă secțiune de program.

Deoarece un simbol (asociat unei date sau instrucțiuni) nu are valabilitate decît în cadrul secțiunii în care este definit, extinderea valabilității lui și în cadrul altor secțiuni ale programului necesită utilizarea directivelor DEF și REF.

*Directiva* DEF – format :

[*etichetă*] DEF $s_1[,s_2] \ldots [,s_n]$ – precizează faptul că simbolurile $s_i$ definite într-o secțiune $S$ pot fi utilizate (referite) și în alte secțiuni diferite de $S$ ; simbolurile $s_i$ se numesc *definiri externe*.

*Directiva* REF – format :

[*eticheta*] REF $s_1[,s_2] \ldots [,s_n]$ – precizează faptul că simbolurile $s_i$ sînt utilizate într-o secțiune $S$, însă sînt definite în alte secțiuni diferite de $S$ ; simbolurile $s_i$ se numesc *referiri externe*.



Scanned with OKEN Scanner

Liniile sursă în care figurează directivele DEF și REF trebuie să preceadă, în text, liniile în care sînt definite sau referite (utilizate) simbolurile $s_i$.

Utilizarea directivelor REF și DEF este suficientă numai în cazul în care secțiunile în care figurează corespund unui singur modul sau unor module incluse în același segment.

10.1.5.2. *Comunicarea între segmente.* Între segmentele unui program pot fi realizate două tipuri de comunicări :

a) într-un segment de program $S_1$ sînt utilizate date definite într-un alt segment $S_2$ (care poate fi un alt segment de program, segmentul de date asociat segmentului de program $S_1$ sau un segment de date comune) ;

b) dintr-un segment de program $S_1$ se cere încărcarea în memorie și execuția unui alt segment de program $S_2$.

a) Un segment de program $S_1$ poate avea acces la datele unui alt segment $S_2$ (segmentele $S_1$ și $S_2$ fiind situate pe aceeași ramură) prin intermediul unui registru de bază – diferit de registrul R14 – care trebuie specificat în fiecare secțiune de program ce corespunde segmentului $S_1$ și care utilizează date diferite în segmentul $S_2$. Indicarea acestui registru de bază se realizează prin intermediul *directivei* USING, care are următorul format :

[*eticheta*] USING,*n* *r*

unde *n* este numele segmentului $S_2$, iar *r*, numărul registrului de bază care îi este afectat ($9 \leqslant r < 14$). În acest caz, *n* trebuie să fie definit, în secțiunea de program ce corespunde segmentului $S_1$, ca referire externă de tip nume de segment. Acest lucru se realizează prin intermediul *directivei* SEGREF, care are următorul format :

[*etichetă*] SEGREF $n_1[,n_2] \ldots [,n_m]$

unde $n_i$ sînt nume de segmente.

Deci, prin intermediul directivei SEGREF se declară ca referiri externe numele segmentelor la ale căror date se fac referiri într-o secțiune de program.

Observații. 1. O directivă SEGREF în care este citat un segment $S$, trebuie să preceadă, în cadrul secțiunii, directivele USING care citează acest segment.

2. În cadrul unei secțiuni pot apărea mai multe directive USING care citează același segment ; în acest caz apariția unei directive USING anulează efectul celei precedente.

3. În cazul în care segmentul $S_2$ este segmentul de date asociat segmentului $S_1$, nu este necesară prezența directivei SEGREF, ci numai a directivei USING, în care numele segmentului $S_2$ se indică prin expresia SD'%'.

4. Referirea la datele segmentului $S_2$ trebuie să fie precedată de încărcarea în memorie a acestui segment (dacă nu este încărcat) și încărcarea în registrul *r*, specificat în



Scanned with OKEN Scanner

directiva USING, a adresei de bază a acestuia. Încărcarea în memorie a unui segment este indicată prin *macroinstrucțiunea* LDSG, al cărei format este

[*eticheta*] LDSG *n*

unde *n* este numele segmentului $S_2$ (încărcarea în memorie a segmentului $S_2$ este necesară numai dacă acesta este un segment de program, deoarece segmentele de date și de date comune sînt încărcate odată cu segmentul de program căruia îi sînt asociate). Încărcarea adresei de bază a segmentului $S_2$ se realizează prin instrucțiunea

[*eticheta*] LD4,*r* *n*

unde *r* este numărul registrului de bază citat în directiva USING, iar *n*, numele segmentului (în cazul în care segmentul $S_2$ este un segment de date, *n* se indică prin expresia SD'%').

5. Datele la care se fac referiri trebuie declarate, prin directivele REF și DEF, referiri externe, în secțiunea ce corespunde segmentului $S_1$, și definiri externe, în secțiunea ce corespunde segmentului $S_2$.

b) Încărcarea în memorie și execuția unui segment de program $S_2$ pot fi cerute, din cadrul unui segment de program $S_1$, în mai multe moduri :

1. *Prin intermediul macroinstrucțiunii* CALL, al cărui format este următorul :

[*eticheta*] CALL [,L] *p*

unde *p* este numele unui punct de intrare în segmentul $S_2$. Simbolul *p* trebuie declarat în segmentul $S_1$ ca referire externă de tip punct de intrare, prin intermediul *directivei* ENTREF, al cărei format este următorul :

[eticheta] ENTREF $p_1$ [,$p_2$] ... [,$p_n$]

unde $p_i$ reprezintă etichetele instrucțiunilor ce constituie puncte de intrare în segmentul $S_2$. În segmentul $S_2$ simbolul *p* trebuie declarat — prin intermediul cartelei ENTSG — ca punct de intrare și prin intermediul directivei DEF — ca definire externă (aceasta numai dacă simbolul *p* nu reprezintă eticheta unei directive CSECT).

Prin execuția macroinstrucțiunii CALL se realizează încărcarea în memorie a segmentului $S_2$ (dacă acesta nu este deja încărcat) și inițierea execuției acestuia, începînd cu instrucțiunea a cărei etichetă este *p*. Prezența și, respectiv, absența argumentului *L* al macroinstrucțiunii CALL indică un salt cu legătură (cu încărcarea în registrul R8 a adresei de revenire în segmentul $S_1$) și, respectiv, salt fără legătură.

Macroinstrucțiunea

CALL,L *p*

este transformată de asamblor în secvența

BSG SE'*p*'
LD4,14 %

iar macroinstrucțiunea

CALL *p*

în instrucțiunea

BSG SE'*p*'

*Instrucțiunea* BSG are următorul format :

[*etichetă*] BSG,[*e*] SE'*p*'



Scanned with OKEN Scanner

unde *e* reprezintă o expresie aritmetică a cărei valoare poate fi 0 sau 1, iar *p*, un punct de intrare în segment. Prin execuţia instrucţiunii BSG se realizează : verificarea dacă segmentul ce conţine punctul de intrare *p* este în memorie şi, eventual, încărcarea lui ; saltul fără legătură la punctul de intrare în segment, dacă valoarea expresiei *e* este 0, sau saltul la punctul de intrare în segment şi încărcarea în registrul R8 a adresei instrucţiunii care urmează instrucţiunii BSG, dacă valoarea expresiei *e* este 1.

Programul 10.1

```
$       JOB P101,AN:ISO5,PN:FELIX
$       SEG FORT1
$       COMPILE FORTRAN
C
C   PROGRAMUL REALIZEAZA CITIREA ELEMENTELOR A DOUA
C   MATRICI A SI B (IN PROGRAMUL PRINCIPAL),ADUNAREA
C   ACESTORA(IN SUBPROGRAMUL FOR2),TIPARIREA MATRICEI
C   REZULTAT D(IN SUBPROGRAMUL FOR2),DETERMINAREA
C   SUMEI ELEMENTELOR LINIEI 4 A MATRICEI D (IN SUB-
C   PROGRAMUL FOR3),TIPARIREA SUMEI(IN PROGRAMUL
C   PRINCIPAL),DIFERENTA DINTRE MATRICELE A SI B(IN
C   SUBPROGRAMUL FOR4),TRANSPUSA MATRICEI REZULTAT
C   E(IN SUBPROGRAMUL FOR5) SI TIPARIREA ACESTEIA(IN
C   SUBPROGRAMUL FOR6)
C
$ SEGMENT BLOC1+BLOC2,BLOC3
      COMMON/BLOC1/A(5,5)/BLOC2/B(5,5)/BLOC3/C(5,5)
      DIMENSION D(5,5)
      READ(105,1)A,B
    1 FORMAT(25F3.1)
      CALL FOR2
      WRITE(108,2)C
    2 FORMAT(' ','MATRICE C = ',25F4.1)
      DO 4 I=1,5
      DO 3 J=1,5
      D(I,J)=C(I,J)
    3 CONTINUE
    4 CONTINUE
      CALL FOR3(D,4,S)
      WRITE(108,5)S
    5 FORMAT(' ','SUMA ELEMENTELOR LINIEI 4 =',F5.1)
      CALL FOR4
      STOP
      END
$       SEG FORT2
$       COMPILE FORTRAN
$ SEGDATA(DATA) BLOC1+BLOC2,BLOC3
      SUBROUTINE FOR2
      COMMON/BLOC1/A(5,5)/BLOC2/B(5,5)/BLOC3/C(5,5)
      DO 1 I=1,5
      DO 2 J=1,5
      C(I,J)=A(I,J)+B(I,J)
    2 CONTINUE
    1 CONTINUE
      RETURN
      END
$       SEG FORT3
$       COMPILE FORTRAN
      SUBROUTINE FOR3(D,I,S)
      DIMENSION D(5,5)
      S=0.
```

Instrucţiunea

LD4, 14 %

are ca efect încărcarea, din tabela segmentelor, a adresei de bază a segmentului care o conţine. Utilizarea acestei instrucţiuni este obligatorie ori de cîte ori se trece la execuţia unui segment.

2. *Prin intermediul instrucţiunii* BSG, care permite, de asemenea, salturi, cu sau fără legătură, la punctul de intrare în segmentul $S_2$. Problemele care se pun în acest caz sînt asemănătoare celor prezente la punctul 1. În plus, dacă valoarea expresiei *e* este 1, la revenirea din segmentul apelat, prima instrucţiune care trebuie executată este LD4,14 %.



Scanned with OKEN Scanner

```
      DO 1 J=1,5
      S=S+D(I,J)
    1 CONTINUE
      RETURN
      END
.          SEG FORT4
.          COMPILE FORTRAN
* SEGMENT BLOC1+BLOC2,BLOC4
      SUBROUTINE FOR4
      COMMON/BLOC1/A(5,5)/BLOC2/B(5,5)/BLOC4/E(5,5)
      DO 2 I=1,5
      DO 1 J=1,5
      E(I,J)=A(I,J)-B(I,J)
    1 CONTINUE
    2 CONTINUE
      CALL FOR5
      CALL FOR6
      RETURN
      END
.          SEG FORT5
.          COMPILE FORTRAN
* SEGMENT BLOC1+BLOC2,BLOC4
      SUBROUTINE FOR5
      COMMON/BLOC4/E(5,5)
      DO 2 I=1,5
      DO 1 J=1,5
      AUX=E(I,J)
      E(J,I)=AUX
      E(I,J)=E(J,I)
    1 CONTINUE
    2 CONTINUE
      RETURN
      END
.          SEG FORT6
.          COMPILE FORTRAN
* SEGMENT BLOC4
      SUBROUTINE FOR6
      COMMON/BLOC4/E(5,5)
      WRITE(108,1)E
    1 FORMAT(' ','MATRICE E = ',25F4.1)
      RETURN
      END
.          TREE FORT1+(BLOC1+BLOC2)+BLOC3(FORT2,FORT3,FORT4+BLOC4(FORT5,*
.                    FORT6))
.          LINK
.          RUN
222333444555666777888999000111222333444555666777888999000111222333444555666
111222333444555666777888999000111222333444555666777888999000111222333444555
           EOJ
```

3. *Prin intermediul instrucțiunii* BRU *sau* BAL, care permit efectuarea unui salt — cu sau fără legătură — la punctul de intrare în segmentul $S_2$. În acest caz punctul de intrare trebuie definit ca referire externă, în segmentul $S_1$, și ca definire externă, în segmentul $S_2$. De asemenea, saltul trebuie să fie precedat de încărcarea segmentului $S_2$ în memorie și încărcarea, în registrul citat în directiva USING, a adresei de bază a acestui segment; acest lucru implică citarea numelui segmentului $S_2$ într-o directivă SEGREF. Programele 10.1, 10.2 și 10.3 exemplifică segmentarea programelor FORTRAN, COBOL și ASSIRIS.



Scanned with OKEN Scanner

## Programul 10.2

```
    JOB P102,AN:ISO5,PN:FELIX
    COMPILE COBOL
*
*  PROGRAMUL REALIZEAZA:
*    - VALIDAREA DATELOR ARTICOLELOR FISIERULUI PE CARTELE FCART (IN
*      SUBPROGRAMUL VALID);
*    - CREAREA DIN ARTICOLELE FISIERULUI FCART A FISIERULUI SECVENTIAL
*      INDEXAT FINDEX(IN SECTIUNEA CREARE);
*    - TRANSFERUL ARTICOLELOR FISIERULUI FINDEX IN FISIERUL FMAN,ORDO-
*      NAREA LOR DUPA VALORILE ALTEI CHEI DECIT CHEIA DE IDENTIFICARE
*      SI CREAREA FISIERULUI FBAND;
*    - LISTAREA ARTICOLELOR FISIERULUI FBAND(IN SECTIUNEA LISTARE).



      *
       ID DIVISION.
       PROGRAM-ID. PROG102.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FCART  ASSIGN TO SYSIN.
           SELECT FMAN ASSIGN TO BRD.
           SELECT FBAND ASSIGN TO CRT.
           SELECT FINDEX ASSIGN TO ARD
                         ORGANIZATION INDEXED
                         ACCESS SEQUENTIAL
                         RECORD KEY IS IX.
       DATA DIVISION.
       FILE SECTION.
       FD  FCART  RECORDING F
                  LABEL RECORD OMITTED.
       01  CART.
         02 X       PIC 9(5).
         02 Y       PIC A(10).
         02 Z       PIC 9(7).
         02 W       PIC X(25).
       FD  FINDEX  RECORDING F
                   LABEL RECORD STANDARD
                   BLOCK 20 RECORDS.
       01  IART.
         02 IY      PIC A(20).
         02 IX      PIC 9(5).
       SD  FMAN    DATA RECORD MART.
       01  MART.
         02 MY      PIC A(20).
         02 MX      PIC 9(5).
       FD  FBAND  RECORDING F
                  LABEL RECORD STANDARD
                  BLOCK 10 RECORDS.
       01  BART.
         02 BI      PIC A(20).
         02 BX      PIC 9(5).
       WORKING-STORAGE SECTION.
       77  K        PIC 9.
       PROCEDURE DIVISION.
       RADACINA SECTION.
       R1.
           OPEN INPUT FCART OUTPUT FINDEX.
```

```
R2.
     READ FCART AT END
                        CLOSE FCART FINDEX
                        GO TO R3.
     MOVE 0 TO K
     CALL VALID USING CART  K
     IF K = 1 DISPLAY ' ER ' CART GO TO R2.
        GO TO CREARE.
R3.  SORT FMAN ASCENDING KEY MY
               INPUT PROCEDURE IS INTRARE
               GIVING FBAND
     GO TO L1.
INTRARE SECTION.
I1.
     OPEN INPUT FINDEX.
I2.
     READ FINDEX AT END GO TO I3.
     RELEASE MART FROM IART GO TO I2.
I3.
     CLOSE FINDEX.
CREARE SECTION 60.
C1.
     MOVE X TO IX
     MOVE Y TO IY


     WRITE IART INVALID KEY DISPLAY ' ERCARE ' IX.
     GO TO R2.
LISTARE SECTION 70.
L1.
     OPEN INPUT FBAND.
L2.
     READ FBAND AT END CLOSE FBAND STOP RUN.
     DISPLAY BI ' ' BX GO TO L2.
   COMPILE COBOL
ID DIVISION.
PROGRAM-ID. VALID.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
LINKAGE SECTION.
01  K              PIC 9.
01  CART.
  02 X       PIC 9(5).
  02 Y       PIC A(10).
PROCEDURE DIVISION USING CART K.
1.
     IF X NOT NUMERIC MOVE 1 TO K.
     IF Y NOT ALPHABETIC MOVE 1 TO K.
2.
     EXIT PROGRAM.
   TREE PROG1X00(VALIDX00,PROG1X11,PROG1X21,RXDRACI(RXTDINIT,
*            RXTDUN,RXTDDE,RXTDTR,RXTDQU,RXTDERT))
*    LINK
*    INIT DVT:RD,VS:RD1MAN
*    INIT DVT:MT,VS:MT1MAN
*    ASSIGN A,DVT:RD,VS:RD1MAN
*    LABEL A,FN:'FINDEX',AM:ANY,BZ:5
*    ASSIGN B,DVT:RD,VS:RD1MAN
*    LABEL B,FN:'FMAN',AM:ANY,BZ:8
*    ASSIGN C,DVT:MT,VS:MT1MAN
*    LABEL C,FN:'FBAND'
*    RUN
*1111CCCCCCCCC1111111
*2222BBBBBBBBB2222222
*3333AAAAAAAAA4444444
*EOF
*    EOJ
```

Programul 10.3

```
*       JOB P103,AN=IS05,PN=FELIX
*       SEG ALFA
*       COMPILE ASSIRIS
        CSECT     0
        DEF       Z,X,Y
Z       DATA,4,4  -15,21,30
X       RES,6     1
Y       RES,4     1
SUB1    CSECT     P
        REF       Z,X,Y
        ENTREF    INT1,INT2
        USING,SD'X'  12
START   LD4,12    SD'X'
        LD6,0     X
        AD4,0     Z+4

        AD4,0     Z+8
        CALL,L    INT1
        ST6,6     X
        CALL,L    INT2
        ST6,7     Y
        TERN
        END       START
*       SEG BETA
*       ENTSG INT1,INT2
*       COMPILE ASSIRIS
SUB2    CSECT     P
        DEF       INT1,INT2
INT1    LD4,14    X
        LD4,6     0
        AD4I,0    X'FF'
        BRU       *32
INT2    LD4,14    X
        LD4,7     0
        AD4I,7    X'0101'
        BRU       *32
        END
*       LINK
*       RUN UD:CP
*       EOJ
```

## 10.2. UTILIZAREA COMBINATĂ A SUBPROGRAMELOR ASSIRIS, COBOL, FORTRAN

Utilizarea în cadrul aceluiași program a unor subprograme scrise în limbaje diferite (ASSIRIS, COBOL, FORTRAN) este posibilă deoarece modulele obiect au un format standard. Problema care se pune în acest caz este aceea a alegerii limbajului cel mai adecvat pentru scrierea diferitelor părți (secțiuni) ale programului (vezi cap. 1). Capitolele anterioare au relevat, în cazul limbajelor ASSIRIS, COBOL și FORTRAN, facilitățile pe care le oferă. Astfel, programele scrise în limbajul ASSIRIS permit accesul la toate resursele sistemului ; în schimb, utilizarea acestui limbaj presupune o cunoaștere aprofundată a sistemului de calcul, iar programele nu sînt transferabile. Programele scrise în limbajele COBOL și FORTRAN au o anumită independență față de sistemul de calcul (în general, aceste programe sînt transferabile), însă nu permit accesul la toate resursele acestuia, modelul de organizare a memoriei fiind „ pe caractere", în cazul limbajului COBOL, și „pe cuvinte", în cazul limbajului FORTRAN.

Structurarea unui program în unități independente, scrise în limbaje diferite, impune ca una dintre acestea să aibă, în raport cu celelalte, un rol central. Această unitate, numită *program principal*, trebuie să asigure, în principiu, monitorizarea execuției întregului program.

În cele ce urmează sînt prezentate, succint, problemele de interfață care se pun în cazurile în care programul principal este scris în limbajele COBOL, FORTRAN și ASSIRIS.

10.2.1. **Apelarea în programe COBOL a subprogramelor FORTRAN și ASSIRIS.** Transferul parametrilor se realizează prin adrese, compila-



Scanned with OKEN Scanner

torul COBOL generînd pentru instrucţiunea CALL (de apel al subprogramului) o secvenţă de instrucţiuni în limbaj-maşină care, prin execuţie, va realiza :

– alcătuirea unei tabele cu adresele parametrilor actuali (pentru fiecare parametru, în tabelă există un cuvînt-memorie care conţine : în poziţiile binare 8–31 adresa acestuia, iar în poziţiile binare 4–7 lungimea – care este indicată însă numai dacă este mai mică decît 16), tabelă ce va fi inclusă în modulul de date ce corespunde programului principal ;

– încărcarea în registrul R2 a adresei acestei tabele ;

– încărcarea în registrul R8 a adresei de revenire ;

– saltul la punctul de intrare specificat în instrucţiunea CALL ;

– încărcarea în registrul R14 a adresei de bază a segmentului de program în care se află modulul de program ce corespunde programului principal.

În cazul apelării unui subprogram FORTRAN (este posibilă numai apelarea subprogramelor de tip SUBROUTINE) o atenţie deosebită trebuie să se acorde descrierii parametrilor atît în programul principal cît şi în subprogram. Descrieri echivalente ale parametrilor se pot realiza utilizînd în descrierea parametrilor actuali (din programul principal) opţiunile COMP cu PIC S9 (9), COMP-1 şi COMP-2, iar în descrierea parametrilor formali (din subprogram), declaraţiile de tip (explicite sau implicite) INTEGER, REAL şi respectiv DOUBLE PRECISION, în cazul în care parametrii reprezintă date numerice în cod complementar cuvînt, virgulă mobilă simplă precizie şi, respectiv, virgulă mobilă dublă precizie. De asemenea, parametrilor care reprezintă date nenumerice (şiruri de caractere) trebuie să li se asocieze, prin descriere, zone de memorie care să reprezinte un număr întreg de cuvinte-memorie ; în programul principal acest lucru se poate realiza utilizînd clauza SYNCHRONIZED (un parametru ce reprezintă o dată nenumerică de lungime mai mare decît patru, trebuie definită în subprogram ca tablou ; de exemplu, descrierile

```
77 A          PIC X (21) SYNCH LEFT   în COBOL şi
DIMENSION A (6)                       în FORTRAN
```

sînt echivalente).

În programul principal COBOL, apelul subprogramului se realizează prin instrucţiunea CALL (în care este citat numele subprogramului din instrucţiunea SUBROUTINE sau numele unui punct de intrare definit de instrucţiunea ENTRY), iar revenirea din subprogram prin instrucţiunea RETURN.

În cazul apelării unui subprogram ASSIRIS, în instrucţiunea CALL este citat numele unui punct de intrare definit prin cartela ENTSG şi, eventual, în directiva DEF (dacă numele punctului de intrare nu este eticheta directivei CSECT).



Scanned with OKEN Scanner

Parametrii citaţi în instrucţiunea CALL nu sînt definiţi în subprogramul ASSIRIS, adresarea lor realizîndu-se prin intermediul tabelei de adrese generată de compilatorul COBOL, a cărei adresă este furnizată de registrul R2. Pentru a simplifica adresarea, această tabelă poate fi transferată într-o zonă rezervată în subprogramul ASSIRIS.

Revenirea din subprogram, în programul principal, se poate realiza, de exemplu, printr-o instrucţiune BRU *32 sau BRU 0,8.

Observaţii. Modulele de program ce corespund subprogramelor FORTRAN şi ASSIRIS pot fi incluse în segmentul de program care corespunde programului principal sau pot forma segmente de program distincte (dacă acest lucru este indicat în cartela SEG). În ultimul caz, punctul de intrare în subprogramul ASSIRIS trebuie să conţină o instrucţiune de încărcare, în registrul R14, a adresei de bază a segmentului în care este inclus modulul de program ce corespunde subprogramului.

**10.2.2. Apelarea în programe FORTRAN a subprogramelor COBOL şi ASSIRIS.** Comunicarea parametrilor se realizează prin intermediul tabelei de adrese generată de compilatorul FORTRAN, care se deosebeşte de tabela de adrese generată de compilatorul COBOL prin următoarele :

— primul cuvînt-memorie conţine numărul de locaţii ocupate de tabelă adică $(n+1)*4$, unde $n$ reprezintă numărul parametrilor actuali ;

— următoarele $n$ cuvinte-memorie conţin, fiecare, în ultimele trei locaţii adresa parametrului, iar în prima locaţie un cod, cu valori între $00_{16}$ şi $0C_{16}$, a căror semnificaţie este :

$02_{16}$ — variabilă de tip întreg ;
$03_{16}$ — variabilă de tip întreg, lungime dublă ;
$04_{16}$ — variabilă de tip real precizie simplă ;
$05_{16}$ — variabilă de tip real precizie dublă ;
$06_{16}$ — variabilă de tip complex ;
$07_{16}$ — variabilă de tip complex, lungime dublă ;
$08_{16}$ — variabilă logică ;
$09_{16}$ — variabilă logică, lungime dublă ;
$0A_{16}$ — constantă Hollerith ;
$0B_{16}$ — etichetă transmisă ca parametru ;
$0C_{16}$ — nume de subprogram transmis ca parametru.

Secvenţa de instrucţiuni în limbaj-maşină generată prin traducerea instrucţiunii CALL, de apel al subprogramului, realizează operaţiile :

— încărcarea în registrul R2 a adresei tabelei parametrilor ;

— saltul la punctul de intrare în subprogram cu memorarea, în registrul R8, a adresei de revenire ;

— încărcarea în registrul R14 a adresei de bază a segmentului de program ce corespunde programului principal.



Scanned with OKEN Scanner

În cazul apelării unui subprogram COBOL, numele punctului de intrare (indicat în instrucțiunea CALL) este numele subprogramului, din paragraful PROGRAM-ID, iar revenirea din subprogram este indicată prin instrucțiunea EXIT PROGRAM. Deoarece, la traducerea subprogramului, compilatorul COBOL generează o cartelă SEG, modulul de program ce corespunde programului principal trebuie inclus într-un segment distinct (prin intermediul unei cartele SEG). În schimb, modulul de program ce corespunde unui subprogram ASSIRIS poate fi inclus în segmentul de program ce corespunde programului principal FORTRAN.

Accesul la datele segmentelor de date comune definite în programele FORTRAN prin directivele * SEGMENT sau * SEGDATA poate fi realizat din subprogramul ASSIRIS, de exemplu, prin intermediul secțiunilor de date comune (modulele de date comune ce corespund acestor secțiuni, pot forma segmente de date comune distincte, sau pot fi grupate în segmente de date comune, în același mod ca și în cazul utilizării directivelor * SEGMENT sau * SEGDATA). Variabilele grupate în blocurile de tip „common", utilizate în subprogramul ASSIRIS, trebuie declarate definiri externe, iar pentru adresarea lor, secțiunilor de date comune în care sînt definite, trebuie să li se afecteze un registru de bază. (În cazul segmentării prin directiva * SEGMENT, registrele de bază afectate segmentelor de date comune sînt R9 – R12, în ordinea în care segmentele de date comune figurează în directivă ; în cazul utilizării directivei * SEGDATA, segmentului de date generat îi este afectat registrul R9, iar segmentelor de date comune, registrele R10 – R12).

10.2.3. **Apelarea în programe ASSIRIS a subprogramelor COBOL și FORTRAN.** Realizarea interfeței ASSIRIS – COBOL și ASSIRIS – FORTRAN necesită cunoașterea secvențelor de instrucțiuni generate de compilatoarele COBOL și FORTRAN în punctele de intrare citate în instrucțiunile de apel al subprogramelor, precum și în punctele de ieșire din subprograme. Astfel, compilatorul COBOL generează o secvență de instrucțiuni, care, la execuție, va realiza :

– încărcarea în registrul R14 a adresei de bază a segmentului de program ce conține modulul de program care corespunde subprogramului, salvarea conținutului registrului TALLY, transferul în subprogram a tabelei cu adresele parametrilor – utilizînd conținutul registrului R2, încărcarea în registrele R13 și R14 a adreselor de bază a segmentelor de program și de date ce corespund subprogramului, poziționarea pe unu a măștilor de depășire – în punctul de intrare ;

– restaurarea conținutului registrelor R8 – R12 și a registrului TALLY – în punctul de ieșire.



Scanned with OKEN Scanner

## Programul 10.4

```
      JOB P104,AN:1505,PN:FELIX
      COMPILE COBOL
*
*     PROGRAMUL REALIZEAZA:
*      - CITIREA DE PE CARTELE A ELEMENTELOR MATRICEI A(N,N) 2<N<20(IN
*        PROGRAMUL PRINCIPAL COBOL) SI TIPARIREA ELEMENTULUI MAXIM;
*      - N ESTE FURNIZAT PRIN CARTELA OPTION SI ESTE TRANSFERAT DIN ZONA
*        DE OPTIUNI IN PROGRAMUL PROGRAMUL PRINCIPAL COBOL PRIN INTERME-
*        DIUL SUBPROGRAMULUI ASSIRIS;
*      - ELEMENTUL MAXIM ESTE DETERMINAT IN SUBPROGRAMUL FORTRA.
*
       ID DIVISION.
       PROGRAM-ID. COBO.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  N                       PIC 99.
       01  NI                      PIC S9(5) COMP.
       01  INDICI.
           02 I                    PIC S99 COMP.
           02 J                    PIC S99 COMP.
       01  LINIE.
           02 EL                   PIC S9V99
                                   OCCURS 2 TO 20 DEPENDING N.
       01  MATRIC.
           02 L                    OCCURS 20.
              03 E                 OCCURS 20 COMP-1.
       01  MAXIM                   COMP-1.
       01  MAXIM-EDIT              PIC +++.99.
       PROCEDURE DIVISION.
       APEL.
           CALL ASSIRIS USING N
           PERFORM CITIRE THRU CITIRE-EXIT
                   VARYING I FROM 1 BY 1 UNTIL I > N


           MOVE N TO NI
           CALL FORTRA  USING NI, MATRIC, MAXIM
           MOVE MAXIM TO MAXIM-EDIT
           DISPLAY ' ELEMENT MAXIM = ' MAXIM-EDIT
           STOP RUN.
       CITIRE.
           ACCEPT LINIE
           PERFORM TRANS VARYING J FROM 1 BY 1 UNTIL J > N
           GO TO CITIRE-EXIT.
       TRANS.
           MOVE EL (J) TO E (I, J).
       CITIRE-EXIT.
           EXIT.
*        ENTSG ASSIRIS
*        COMPILE ASSIRIS
         CSECT
         DEF       ASSIRIS
ADRN     RES,4     1
ASSIRIS  LD4,14    %
         LD4,0     *8
         ST4,0     ADRN
         LD4,0     *60
         AD4I,U    36
         LD4I,1    2
         MVSR,0    *ADRN
         BRU       *32
         END
        COMPILE FORTRAN
      SUBROUTINE FORTRA(N,MATRIC,MAXIM)
      REAL MAXIM,MATRIC(N,N)
       M=N+1
       MAXIM=MATRIC(1,1)
       DO 2 I=1,N
       DO 1 J=2,M
       IF (MAXIM.GE.MATRIC(I,J-1))GO TO 1
       MAXIM=MATRIC(I,J-1)
```

```
                    1 CONTINUE
                    2 CONTINUE
                      RETURN
                      END
                .          LINK
                .          OPTION CS'04'
                .          RUN
                0001000200030004
                0010002000300040
                0100020003000400
                1000200030004000
                .          EOJ
```

Programul 10.5

```
.          JOB P105,AN:ISO>,PN:FELIX
.          SEG FORT
.          COMPILE FORTRAN
C
C           PROGRAMUL REALIZEAZA:
C            - CITIREA DE PE O CARTELA A ELEMENTELOR VECTORULUI V(N)-N=10,
C              DETERMINAREA ELEMENTELOR NULE SI TIPARIREA LOR,DETERMINAREA
C              SUMEI PRIMELOR N ELEMENTE-N=6-SI TIPARIREA ACESTEIA.
C            - NUMARUL ELEMENTELOR NULE ESTE DETERMINAT IN SUBPROGRAMUL COBO.
C              IAR SUMA PRIMELOR N ELEMENTE.IN SUBPROGRAMUL ASSIRIS.
C
* SEGMENT
      COMMON/BLOC/V(10)
      DIMENSION VECTOR(10),ZERO(10)
      INTEGER ZERO
      READ(105,1)V
    1 FORMAT(10F3.1)


      DO 2 I=1,10
    2 VECTOR(I)=V(I)
      K=0
      CALL COBO(K,VECTOR,ZERO)
      IF(K.EQ.0)GO TO 4
      WRITE(108,3)(ZERO(I),I=1,K)
    3 FORMAT(' ','ELEMENTELE NULE SINT:',10('E(',I2,') '))
      GO TO 6
    4 WRITE(108,5)
    5 FORMAT(' ','VECTORUL NU ARE ELEMENTE NULE')
      GO TO 9
    6 CALL ASSIR(SUMA,6,&7)
    7 WRITE(108,8)SUMA
    8 FORMAT(' ','SUMA PRIMELOR 6 ELEMENTE ESTE:',F5.1)
    9 STOP
      END
         COMPILE COBOL
       ID DIVISION.
       PROGRAM-ID. COBO.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       DATA DIVISION.
       LINKAGE SECTION.
       01  K              PIC S9(5) COMP.
       01  VEC.
         02 VECTOR        OCCURS 10
                          INDEXED BY I.
           03 E           COMP-1.
       01  ZER.
         02 Z             OCCURS 10 PIC S9(5) COMP.
       PROCEDURE DIVISION USING K, VEC, ZER.
       1.
```

```
            SET I TO 1.
        2.
            SEARCH VECTOR VARYING I
                   AT END GO TO 3
                   WHEN E (I) = 0 ADD 1 TO K
                                  SET Z (K) TO I
                                  SET I UP BY 1
                                  GO TO 2.
        3.
            EXIT PROGRAM.
.           SEG ASSI
.           ENTGG ASSIR
.           COMPILE ASSIRIS
ASSIRIS     CSECT     P
            DEF       ASSIR
            SEGREF    BLOC
            USING,BLOC 9
REG         RES,4     14
ADRSUMA     RES,4     1
ADRSASE     RES,4     1
ADROPT      RES,4     1
ASSIR       LD4,14    %
            STM,13    REG
            LD4,9     BLOC
            LD4I,3    12
            MVSR,2    ADRSUMA
            LD4I,0    0
            LD4I,1    0
            LD4,2     *ADRSASE
ADD         ADF4,0    *BLOC,1
            SB4I,2    1
            BZ        REV
            AD4I,1    4
            BRU       ADD
REV         ST4,0     *ADRSUMA
            LDM,13    REG
            BRU       *ADROPT


            END
.           TREE FORT+BLOC(COBOXXOO,ASSI)
.           LINK
.           RUN
01002003004000005000006000070
.           EOJ
```

Programul 10.6

```
.       JOB P106,AN,1805,PN,FELIX
.       SEG ASS
.       COMPILE ASSIRIS
.
*   PROGRAMUL DETERMINA VALOAREA EXPRESIEI RADICAL(A**2 + B**2)/C**2
*   SI PREGATIREA VALORILOR TERMENILOR SI AI EXPRESIEI PENTRU TIPARIRE.
*   VALOAREA EXPRESIEI ESTE DETERMINATA  IN SUBPROGRAMUL FORTR,IAR
*   PREGATIREA PENTRU TIPARIRE,IN SUBPROGRAMUL COBO.
```



Scanned with OKEN Scanner

```
                CSECT
                REF         FORTR
                ENTREF      COBO
A               DATA,4,4    FS'10,0'
B               DATA,4,4    FS'5,0'
C               DATA,4,4    FS'2,0'
E               RES,4       1
                BOUND       8
AE              RES         11
BE              RES         9
CE              RES         10
EE              RES         14,
TABFORT         DATA,4,4    A
                DATA,4,4    B
                DATA,4,4    C
                DATA,4,4    E
TABCOB          RES,8       1
START           LD4I,2      TABFORT
                BAL,8       FORTR
                LD4I,0      A
                ST4,0       TABCOB
                LD4I,0      AE
                ST4,0       TABCOB+4
                LD4I,2      TABCOB
                CALL,L      COBO
                TERM
                END         START
```

```
     COMPILE FORTRAN
SUBROUTINE FORTR(A,B,C,E)
E=SQRT(A**2+B**2)/C**2
RETURN
END
```

```
     COMPILE COBOL
 ID DIVISION.
 PROGRAM-ID. COBO.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 DATA DIVISION.
 LINKAGE SECTION.
 01  GRUPC.
   02 A              COMP-1.
   02 B              COMP-1.
   02 C              COMP-1.
   02 E              COMP-1.


 01  GRUPE.
   02 AE             PIC +(8).99.
   02 BE             PIC +(6).99.
   02 CE             PIC +(8).99.
   02 EE             PIC +(11).99.
PROCEDURE DIVISION USING GRUPC, GRUPE.
*.
     MOVE A TO AE
     MOVE B TO BE
     MOVE C TO CE
     MOVE E TO EE.
2.
     EXIT PROGRAM.
```

```
    LINK
    RUN UD1:CP
    EOJ
```



Scanned with OKEN Scanner

Secvențele generate de compilatorul FORTRAN realizează :

– încărcarea în registrul R14 a adresei de bază a segmentului de program ce corespunde subprogramului, salvarea conținutului registrelor R9 – R12 (numărul acestora este în funcție de numărul segmentelor de date și de date comune definite în subprogram) și încărcarea în registrele R9 – R12 a adreselor de bază ale segmentelor de date și de date comune (dacă în subprogram sînt definite astfel de segmente), transferul în subprogram a tabelei cu adresele parametrilor, utilizînd conținutul registrului R2 (sînt transferate numai ultimele *n* cuvinte, *n* fiind numărul parametrilor formali) în punctul de ieșire ;

– restaurarea conținutului registrelor R9 – R12 (dacă este cazul) în punctul de ieșire.

Din cele prezentate rezultă că în programul principal ASSIRIS trebuie constituită tabela cu adresele parametrilor, iar adresa acesteia trebuie încărcată, înaintea apelului subprogramului, în registrul R2, iar adresa de retur în registrul R8. De asemenea, punctele de intrare citate în instrucțiunea de apel (BRU, BAL, CALL, BSG) trebuie definite ca referiri externe. Programele 10.4, 10.5 și 10.6 ilustrează cele prezentate în cadrul acestui subcapitol.

Scanned with OKEN Scanner

# 11

# TESTAREA ŞI PUNEREA LA PUNCT A PROGRAMELOR

Scopul acestui capitol este de a prezenta, succint, fazele de testare şi punere la punct care preced implementarea programelor, metodele ce pot fi folosite pentru prevenirea, punerea în evidenţă şi înlăturarea erorilor. Referirile concrete vor viza facilităţile de testare şi punere la punct oferite de sistemul de operare SIRIS 2/3, în cazul programelor ASSIRIS, COBOL şi FORTRAN.

## 11.1. GENERALITĂŢI

*Testarea* trebuie să pună în evidenţă erorile şi constă în execuţia programelor cu seturi convenabile de date de test, pentru care, în general, se cunosc rezultatele ce trebuie obţinute [100]. *Punerea la punct* are ca scop identificarea cauzelor care au determinat apariţia erorilor şi constă în utilizarea unor metode pentru înlăturarea lor. Aceste două faze se succed pînă în momentul în care, în faza de testare, nu mai sînt semnalate erori. Trebuie făcută însă observaţia că sfîrşitul testării nu garantează totuşi o corectitudine absolută a programului, deoarece variantele de date de test utilizate nu validează, în general, funcţionarea corectă a tuturor instrucţiunilor programelor. Utilizarea mai frecventă a unor metode moderne de programare (programare structurată, modulară etc.), transferabilitatea şi portabilitatea programelor au ca efect reducerea probabilităţii de apariţie a erorilor. Cu toate acestea, importanţa fazelor de testare şi punere la punct nu trebuie diminuată, deoarece apariţia erorilor nu poate fi complet eliminată. Depistarea şi corectarea erorilor poate fi costisitoare şi de durată (se apreciază că acestea pot reprezenta aproximativ 50% din timpul de realizare a unui program [100]), dacă



Scanned with OKEN Scanner

în celelalte faze de realizare a programelor — proiectare, elaborare documentaţie, codificare etc. — nu se au în vedere cerinţele testării şi punerii la punct. Întrucît, în practică, testarea şi punerea la punct nu pot fi separate, se poate considera că există factori comuni care le influenţează. Dintre aceştia, cei mai importanţi sînt prezentaţi în continuare.

a) *Documentaţia programului.* Evident că, lipsa unei documentaţii corecte şi complete are implicaţii negative asupra celor două faze şi, în special, asupra punerii la punct. O atenţie deosebită trebuie acordată, de asemenea, lizibilităţii programului. Trebuie precizată aici şi importanţa comentariilor ; este indicat ca pentru fiecare dată (variabilă) să se indice, prin comentarii, semnificaţia, fiecărei componente a programului să-i fie asociată o descriere din care să reiasă funcţia şi algoritmul utilizat, intrările şi ieşirile pe care le produce etc.

Prezenţa comentariilor în textul programului poate ajuta programatorul să-şi reamintească acţiunile ce trebuie întreprinse în timpul testării şi punerii la punct (de exemplu, inserarea în program a unui comentariu care precizează intervalul de valori asociat unei variabile, va constitui un ajutor preţios pentru programator în situaţia în care, în faza de punere la punct, trebuie modificată o secvenţă a programului în care este referită variabila respectivă).

b) *Inserarea în cadrul programelor a unor rutine de trasare* (realizate de utilizator sau ale sistemului de operare) *sau a unor instrucţiuni de punere la punct* care să permită urmărirea controlului execuţiei programului, pot simplifica, de asemenea, testarea şi punerea la punct. Inserarea, explicită sau nu, a acestor rutine şi instrucţiuni în program (care este funcţie de facilităţile de testare şi punere la punct oferite de sistemul de operare), trebuie prevăzută la proiectarea programului, deoarece adăugarea lor ulterioară este mai dificilă, putînd afecta corectitudinea acestuia.

Rutinele de trasare şi instrucţiunile de punere la punct pot avea ca scop :

— tipărirea valorilor curente ale unor date sau rezultate intermediare, pentru a marca execuţia unei instrucţiuni sau subprogram ;

— testarea valorii unei date pentru a vedea dacă aceasta aparţine intervalului de valori specificat în program ;

— memorarea şi tipărirea unor informaţii pentru completarea celor furnizate de vidajul memoriei (de exemplu, în situaţiile în care informaţiile furnizate de vidaj nu sînt suficiente pentru localizarea erorilor, în program poate fi inserată o rutină care, la fiecare execuţie a componentelor acestuia, să memoreze, într-o zonă-memorie, următoarele informaţii : numele componentei, adresa instrucţiunii de apel, valorile parametrilor actuali, conţinutul registrelor, valorile datelor modificate prin execuţie, numărul apelului etc. ; tipărirea acestor informaţii poate fi realizată după apelarea de un număr de ori a componentelor programului).



Scanned with OKEN Scanner

În mod obișnuit, rutinele de trasare și instrucțiunile de punere la punct sînt eliminate din programe după testarea și punerea la punct a acestora, pentru optimizarea timpului de execuție. În anumite situații, este însă avantajos să fie păstrate și după implementarea acestora (de exemplu, în cazul în care programele sînt executate în timp real, cînd există posibilitatea introducerii de date eronate).

În general, programele traducătoare asigură inserarea în cadrul programelor a rutinelor de trasase și a instrucțiunilor de punere la punct de tipul celor menționate.

c) *Modularitatea programelor*, prin avantajele pe care le oferă (claritate conceptuală, elaborare pe părți și în echipe a modulelor, elaborare și testare succesivă a modulelor), poate simplifica, de asemenea, testarea și punerea la punct.

În cazul programelor modulare testarea trebuie să demonstreze corectitudinea fiecărui modul cît și a programului în ansamblu. Deoarece un modul, în general, nu poate funcționa de sine stătător, testara lui independentă nu este întotdeauna concludentă. De aceea, realizarea testării necesită elaborarea unui modul care să simuleze apelarea și contextul în care modulul testat va funcționa în cadrul programului. În ceea ce privește metoda testării, este indicată cea „incrementală", care constă în agregarea progresivă a modulelor pe măsura elaborării și testării lor individuale. Aceasta necesită însă simularea modulelor care nu au fost elaborate și testate. Simularea trebuie să asigure transmiterea / primirea valorilor parametrilor ce condiționează execuția programului în ansamblu.

d) *Stabilirea datelor de test* poate, de asemenea, influența testarea și punerea la punct a programelor. În cazul multor programe, și mai ales a celor complexe, demonstrarea corectitudinii acestora ar necesita utilizarea unui număr foarte mare de variante de date de test, ceea ce ar face ca testarea să fie ineficientă. În astfel de situații, cheia succesului constă în abilitatea programatorului de a selecta variantele de date de test care sînt cele mai relevante pentru problema respectivă.

## 11.2. TIPURI DE ERORI

Se poate considera că un program este corect dacă, prin execuție, furnizează rezultate scontate. Deși noțiunea de corectitudine a unui program include în sine noțiunea de corectitudine a algoritmului, în realitate înseamnă mult mai mult. Aceasta deoarece, chiar dacă algoritmul este stabilit corect din toate punctele de vedere, corectitudinea programului poate fi afectată în etapele care urmează stabilirii algoritmului (scrierea programului, pregătirea pentru execuție și chiar execuția) datorită unor



Scanned with OKEN Scanner

interpretări sintactice greșite la transcrierea programului, opțiuni de compilare sau de editare implementate defectuos, asamblări greșite componentelor programului, defecțiuni hardware sau software ale sistemului de calcul, greșeli de operare etc.

Presupunînd că datele de intrare sînt corecte, se poate afirma că există cinci tipuri de erori : *hardware, software, de operare, de programare* și *de sistem*. În [12] se estimează pentru aceste tipuri de erori următoarele procente medii a frecvențelor lor de apariție : 1%, 2%, 5%, 90% și, respectiv, 2%. Erorile hardware sînt foarte rare și sînt, în general, ușor de detectat. Erorile software au o frecvență de apariție foarte mică, însă sînt, în general, greu de detectat. Ultimele trei tipuri de erori prezintă particularitatea că pot interacționa și din aceeastă cauză este dificil de a li se preciza tipul. Desigur că, în cadrul acestor trei tipuri, frecvența cea mai mare de apariție și cele mai dificile sînt erorile de programare.

În general, erorile hardware și de programare sînt semnalate și comunicate utilizatorului de către diferitele componente ale sistemului de operare, în fazele de pregătire a programelor pentru execuție (compilare, editare de legături, încărcare) sau în faza de execuție a acestora.

În cele ce urmează sînt prezentate erorile semnalate de sistemul de operare SIRIS 2/3, considerînd cazul programelor ASSIRIS, COBOL și FORTRAN.

## 11.3. ERORI SEMNALATE ÎN FAZELE DE PREGĂTIRE A PROGRAMELOR PENTRU EXECUȚIE

Pentru înțelegerea facilităților de testare și punere la punct, prezentarea acestor erori va fi însoțită de o expunere succintă a fazelor de pregătire a programelor pentru execuție, a situațiilor editate de programele sistemului de operare care realizează aceste faze. Ilustrarea celor prezentate este realizată prin intermediul programului anexat la sfîrșitul capitolului.

**11.3.1. Compilarea (asamblarea).** În această fază are loc traducerea programelor sursă (care pot fi înregistrate pe cartele sau în biblioteci de tip sursă) în limbajul-mașină caracteristic sistemului de calcul. În urma compilării rezultă unități de program independente, numite *module obiect*. În principiu, un modul obiect este alcătuit din informații de început de modul, informații de sfîrșit de modul, informații de legătură cu alte module și textul în limbaj-mașină rezultat în urma traducerii (textul obiect). Indiferent de limbajul în care este scris programul sursă, în faza



Scanned with OKEN Scanner

de compilare (asamblare) pot fi generate trei tipuri de module : *module de program*, *module de date* și *module de date comune*. Tipul modulelor, conținutul acestora, numele care le sînt atribuite sînt funcție de limbajul în care este scris programul (vezi cap. 10). Textele obiect sînt generate de către programele traducătoare sub formă de succesiuni de baiți, într-o zonă a memoriei centrale. Deoarece această zonă nu va coincide cu zona de memorie în care va fi încărcat textul în vederea execuției, programul traducător identifică această zonă de memorie cu o zonă fictivă a cărei adresă de origine este 0. De asemenea, pentru a ține evidența adreselor locațiilor zonei fictive în care este înregistrat textul obiect, programul traducător atribuie unei variabile, numită *contor de locații*, valori care reprezintă adresele acestor locații. Anumite valori ale contorului de locații vor fi atribuite simbolurilor (identificatorilor) definite în textul sursă.

În funcție de tipul modulelor generate, textele obiect ce le alcătuiesc pot conține : instrucțiunile în limbaj-mașină ce corespund instrucțiunilor din textul sursă, zonele rezervate datelor (variabilelor) și constantelor definite în textul sursă și, în cazul textelor sursă COBOL și FORTRAN, zonele rezervate implicit de programul traducător pentru generarea diferitelor constante, pentru memorarea unor rezultate intermediare etc. Instrucțiunilor din textul sursă le pot corespunde, prin traducere, una sau mai multe instrucțiuni în limbaj-mașină. Pentru anumite operații (ca de exemplu conversii, pregătirea valorilor datelor pentru tipărire, utilizarea unor funcții trigonometrice etc.), programele traducătoare generează și inserează în textul obiect secvențe de apel al unor subprograme (module) din Biblioteca de Subprograme Standard a sistemului, ce vor fi incluse în program la editarea legăturilor. În secvențele de apel sînt indicate numele punctelor de intrare în aceste subprograme, ce pot fi identificate, în lista programului obiect, după primele două caractere : C%, și R% — în cazul subprogramelor apelate de compilatorul COBOL ; I% și F% — în cazul celor apelate de compilatorul FORTRAN ; S% ; X% și A% în cazul celor apelate de asamblor.

Ansamblul modulelor obținute în urma traducerii unui text sursă, sînt înregistrate pe discul sistem, constituind fișierele FILEDIT și REPEDIT, de unde poate fi, eventual, catalogat într-o bibliotecă de tip BT sau perforat pe cartele. De asemenea, rezultatele compilării sînt consemnate în cadrul unei situații înregistrată pe suportul fișierului de ieșire al sistemului ; această situație conține numele modulelor generate, tipul lor (indicat prin literele : P — modul de program ; D — modul de date ; C — modul de date comune) și lungimea (exprimată în hexazecimal, iar în cazul programelor FORTRAN și în zecimal).



Scanned with OKEN Scanner

Două situații editate de programele traducătoare sînt importante pentru punerea la punct a programelor : lista programului obiect și geografia datelor.

a) *Lista programului obiect.* Conținutul textului obiect (lista programului obiect) poate fi obținut la imprimantă, implicit – în cazul textelor ASSIRIS – sau explicit – în cazul textelor COBOL și FORTRAN (dacă în cartela COMPILE figurează opțiunea OBL). Forma de imprimate a acestei situații (tipărită de programul traducător la sfîrșitul procesului de traducere) în cazul programelor ASSIRIS, FORTRAN și COBOL este prezentată în continuare.

*Lista programului obiect editată de asamblor* este structurată în două părți. *Prima parte a listei* conține liniile textului sursă și textele obiect ce le corespund. Două tipuri de linii pot figura în cadrul listei, al căror format este

$$- \; vvvvvvvv \quad A \quad \left\{ \begin{matrix} S \\ B \\ N \\ \text{ƀ} \end{matrix} \right\} \quad nnnn \quad ss \ldots s,$$

în cazul în care în linia sursă figurează directiva EQU (*ss . . s* reprezintă caracterele ce compun linia sursă ; *nnnn*, numărul de ordine atribuit de asamblor liniei sursă ; S, B, N sau ƀ indică : o eroare de secvență a liniei sursă, linia sursă a fost citită dintr-o bibliotecă, linia sursă nu a fost prelucrată de asamblor – fiind sărită printr-o instrucțiune GO TO sau DO – sau, respectiv, o linie sursă normală ; *vvvvvvvv*, valoarea – în hexazecimal – atribuită simbolului prin directiva EQU ; A indică faptul că această valoare este absolută) ;

$$- \; cccc \quad vvvvvvvv \quad \left\{ \begin{matrix} A \\ X \\ \text{ƀ} \end{matrix} \right\} \quad \left\{ \begin{matrix} S \\ B \\ N \\ \text{ƀ} \end{matrix} \right\} \quad nnnn \quad ss \ldots s,$$

în cazul liniilor sursă în care nu figurează directiva EQU (*cccc* reprezintă valoarea în hexazecimal a contorului de locații ; *vvvvvvvv*, valoarea în hexazecimal a textului obiect generat; ƀ, A sau X indică faptul că această valoare este absolută, translatabilă, sau că în linia sursă respectivă figurează o referire externă).

*A doua parte a listei*, care apare numai dacă în textul sursă sînt definite literale, conține pentru fiecare literal o linie de forma

*cccc vvvvvvvv* A

unde : *cccc* reprezintă valoarea în hexazecimal a contorului de locații, iar *vvvvvvvv*, valoarea în hexazecimal a literalului.



Scanned with OKEN Scanner

*Lista programului obiect editată de compilatorul* COBOL este structurată în două părți, care furnizează imaginea modulului (modulelor) de program și a zonelor rezervate și inițializate de compilator pentru diferite date, constante, tabele.

*În prima parte a listei*, fiecărui paragraf (secțiune) îi este asociată o linie care conține numele acestuia și simbolul care îi este atribuit de compilator, iar fiecărei instrucțiuni COBOL, una sau mai multe linii care indică :

– numărul liniei sursă în care figurează instrucțiunea (coloana LIG. SOURC) ;

– verbul instrucțiunii (coloana CODE VERBE) ;

– valoarea – în hexazecimal – a contorului de locații (coloana CPT. ORDI) ;

– secvența de instrucțiuni în limbaj-mașină generată (instrucțiunile în limbaj-mașină sînt indicate atît în hexazecimal, în coloana INSTRUCTION, cît și conform convențiilor limbajului de asamblare, în coloana INSTRUCTION DECODEE) ; instrucțiunile în limbaj-mașină care sînt urmate de simbolul X, conțin referiri externe ;

– numele simbolice ale operanzilor citați în instrucțiunea COBOL sau stabiliți de compilator (numele simbolice, indicate în coloana SYMBOLIQUE, pot reprezenta : nume de date, nume de prelucrare, numele unor puncte de intrare în subprograme ale utilizatorului sau din BSS, numele unor zone rezervate implicit de compilator pentru împachetarea/despachetarea valorilor unor operanzi sau pentru memorarea rezultatelor intermediare ; numele simbolice atribuite de compilator zonelor pe care le rezervă implicit, sînt formate din caracterele C%ZM sau C%RI, urmate de un număr format din trei cifre).

*În cea de-a doua parte a listei* pentru fiecare zonă inițializată de compilator se indică în hexazecimal valoarea contorului de locații (coloana ADRESSE), lungimea (coloana LONGUEUR) și conținutul (coloana VALEUR). Constantele și tabelele generate în aceste zone depind de textul sursă supus compilării ; semnificația lor este următoarea :

– CONSTANTES – zonă de 50 locații, în care sînt înregistrate diferite constante necesare execuției textului obiect (zona este rezervată și inițializată pentru orice program COBOL) ;

– TDF – zonă ce va constitui tabela de descriere a unui fișier definit în program (compilatorul înregistrează în această zonă parametrii ce definesc fișierul, cunoscuți în momentul compilării) ;

– TMF – zonă de opt locații, în care se va constitui tabela de descriere a unui multifișier definit în program ;

– MASQUES D'EDITION – zonă în care sînt generate constantele necesare editării valorilor datelor ;



Scanned with OKEN Scanner

– TABLE DE TRADUCTION – zonă în care este generată o tabelă de traducere, utilizată pentru realizarea testului de clasă ;

– LITTERAUX – zonă în care sînt generate literalele care apar în diviziunea PROCEDURE ;

– ADRESSES INDIRECTES – zonă necesară adresării cu indexare a operanzilor reprezentați în zecimal-împachetat ;

– TABLE DES SORTIES DE PERFORM – zonă necesară controlului execuției programului, în cazul în care sînt utilizate instrucțiuni PERFORM ;

– VALEUR INITIALE – zonele rezervate datelor definite în program (sau rezervate implicit de compilator pentru registrele speciale TALLY, RETURN-CODE etc.) pentru care s-a cerut inițializare.

Deoarece instrucțiunile în limbaj-mașină sînt grupate în modulul de program, iar zonele inițializate, în modulul de date, valorile contorului de locații reprezintă adresele relative – în cadrul modulului respectiv – ale instrucțiunilor sau ale datelor.

*Lista programului obiect editată de compilatorul* FORTRAN este structurată în trei părți. *Prima parte a listei*, care apare numai dacă în program sînt definite fișiere, conține informații utilizate de către SGF în timpul execuției programului și are următoarea formă :

$$\begin{array}{ccccccc} Y_1 & X_{11} & X_{12} & X_{13} & X_{14} & X_{15} & X_{16} \\ Y_2 & X_{21} & X_{22} & X_{23} & X_{24} & X_{25} & X_{26} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ Y_n & X_{n1} & X_{n2} & X_{n3} & X_{n4} & X_{n5} & X_{n6} \end{array}$$

unde $Y_i$ reprezintă valoarea contorului de locații asociată unei zone de memorie de lungime maximă șase cuvinte iar $X_{ij}$ ($j=1, 2, \ldots, 6$) reprezintă conținutul acestor cuvinte-memorie ($Y_i$ reprezintă succesiuni de patru cifre hexazecimale, iar $X_{ij}$ succesiuni de opt cifre hexazecimale).

Primul cuvînt-memorie conține numărul $N$ de numere FORTRAN asociate fișierelor definite în program, iar următorul cuvînt-memorie, adresa unei zone-articol care este comună tuturor fișierelor definite în program. În continuare, pentru fiecare din cele $N$ numere FORTRAN, apare cîte un dublu cuvînt-memorie care conține : adresa primului cuvînt-memorie ce precede tabela de descriere a fișierului respectiv, în primul cuvînt și numărul FORTRAN afectat acestuia, în al doilea cuvînt. Primele trei duble-cuvinte conțin informații referitoare la fișierele ce utilizează unitățile standard ale sistemului : imprimanta, cititorul de cartele, perforatorul de cartele. Urmează apoi, cuvintele-memorie care corespund tabelelor de descriere a fișierelor definite în program (tabelele de descriere a fișierelor ce utilizează unitățile standard ale sistemului nu sînt inițializate de compilatorul FORTRAN, ele fiind inserate, în textul obiect, la editarea legăturilor).



Scanned with OKEN Scanner

*A doua parte a listei*, care are aceeași formă ca prima parte a listei, conține imaginea zonelor rezervate și inițializate de compilator. Structura posibilă a acestei părți este următoarea :

– zona în care este înregistrat conținutul instrucțiunilor FORMAT definite în program ;

– zona asociată instrucțiunilor NAMELIST definite în program (pentru fiecare instrucțiune NAMELIST este rezervat un dublu cuvînt-memorie care conține numele NAMELIST, urmat de cîte patru cuvinte-memorie pentru fiecare variabilă citată în instrucțiunea NAMELIST, primele două cuvinte-memorie indică numele variabilei, următorul, tipul și adresa variabilei, iar ultimul indică, în cazul tablourilor, dimensiunea tabloului, și un cuvînt-memorie cu conținutul $FFFFFFFF_{16}$ care marchează sfîrșitul acestei zone) ;

– zona asociată variabilelor și tablourilor inițializate prin instrucțiunea DATA ;

– zona care conține tabelele cu adresele parametrilor actuali transmiși subprogramelor apelate din unitatea de program respectivă (vezi 10.2.2) ;

– zona care conține constantele utilizate în program.

*Partea a treia a listei* conține imaginea instrucțiunilor în limbaj-mașină generate de compilator. Fiecărei instrucțiuni FORTRAN îi corespunde, în listă, una sau mai multe linii care indică : numărul liniei sursă în care apare instrucțiunea FORTRAN (în coloana 1) ; valorile contorului de locații (în hexazecimal) asociate instrucțiunilor în limbaj-mașină ce corespund instrucțiunii FORTRAN (în coloana 2) ; conținutul zonelor I, B, Q, X, F și D (în hexazecimal) ale instrucțiunilor în limbaj-mașină (în coloanele 3–8) ; eticheta instrucțiunii FORTRAN (în coloana 9) ; instrucțiunile ASSIRIS ce corespund instrucțiunilor în limbaj-mașină (în coloanele 10–11).

b) *Geografia datelor*. Această situație, tipărită de programele traducătoare dacă în cartela COMPILE figurează opțiunea MAP, furnizează, în principiu, informații referitoare la tipul, lungimea și adresele de implantare a zonelor asociate datelor, în cadrul modulelor obiect.

*În situația tipărită de asamblor*, intitulată TABLE DES SYMBOLES, pentru fiecare simbol se indică : numele, tipul secțiunii în care este definit și numărul de ordine al acestei secțiuni în cadrul textului sursă (tipul secțiunii este precizat prin caracterul P, C, D sau F care indică o secțiune de program, de date comune, de date sau o secțiune fictivă, iar numărul de ordine, printr-un număr format din una sau două cifre) ; valoarea asociată de asamblor simbolului ; tipul simbolului (precizat prin caracterul X, A, Ƀ, D, F, I sau C care indică faptul că simbolul este : referire externă, absolut, translatabil, definire externă translatabilă, fictiv, nede-

finit sau definire externă absolută). Aceste informații sînt tipărite în ordinea alfabetică a simbolurilor, pe secțiuni.

*Geografia datelor tipărită de compilatorul* COBOL conține două tabele, intitulate GEOGRAPHIE DES DONNEES și GEOGRAPHIE DES FICHIERS.

*Primul tabel* precizează implantarea, în cadrul modulului de date, a datelor definite în diviziunea DATA. Fiecărei date (fișier, articol, dată elementară) îi este asociată o linie structurată astfel :

– coloana NLSR, care indică numărul liniei sursă în care data este definită ;

– următoarea coloană, care indică, prin simbolurile FICH, ART sau spațiu, categoria datei – fișier, articol sau dată elementară ;

– coloana NIV, care indică numărul de nivel asociat datei ;

– coloana ADRESSE care conține : valoarea 0 în cazul unui articol definit în secțiunea FILE, sau adresa relativă în cadrul articolului, în cazul unei date ce aparține unui articol definit în secțiunea FILE ; adresa relativă a datei în cadrul modulului de date, în cazul unei date definite în secțiunea WORKING ; valoarea 0, în cazul unei date cu număr de nivel 77 sau 01 definită în secțiunea LINKAGE, sau adresa relativă în cadrul articolului, în cazul datelor ce aparțin articolelor definite în secțiunea LINKAGE ;

– coloana LONGUEUR, care indică în hexazecimal lungimea zonei de memorie rezervată datei ;

– coloana TYPE, care indică tipul și reprezentarea internă a datelor (GR – dată grupată, DE – reprezentare în zecimal-despachetat, DI – reprezentare în zecimal-împachetat, BHW – reprezentare în cod complementar semicuvînt, BW – reprezentare în cod complementar cuvînt, BDW – reprezentare în cod complementar dublu-cuvînt, AN – alfanumerică, NE – dată numerică de editare, FS – reprezentare în virgulă mobilă simplă precizie, FL – reprezentare în virgulă mobilă dublă precizie, ANJ – dată alfanumerică descrisă cu clauza JUSTIFIED, FE – reprezentare în virgulă mobilă format extern) ;

– coloana IDENTIFICATEUR, care indică numele datei.

*Al doilea tabel* conține, pentru fiecare fișier care nu utilizează unitățile standard ale sistemului, o linie structurată astfel : coloana NOM DU FICHIER, care indică numele fișierului ; coloana IDEX care indică identificatorul de exploatare asociat fișierului ; coloana ORG care indică organizarea fișierului (prin simbolurile : SEQ – secvențială, INDX – secvențială-indexată sau DIR – selectivă) ; coloana PRM care indică modul de prelucrare (prin simbolurile : OUTP – creare, INP – consultare, UPDT – actualizare, USIG – fișier de intrare pentru sortare, GIVG – fișier de ieșire pentru sortare) ; coloana ACM, care indică modul



Scanned with OKEN Scanner

de acces utilizat în prelucrare (prin simbolurile : SEQ – acces secvențial, DIR – acces direct) ; coloana RAM, care indică modul de acces la articole (prin simbolurile MOV – în zona-articol sau LOC – în zona-tampon), coloana RCF, care indică formatul articolelor (prin simbolurile FIX – format fix, VAR – format variabil, UND – format nedefinit) ; coloana BFN, care indică, în zecimal, numărul de zone-tampon asociate fișierului ; coloana BFS, care indică, în zecimal, lungimea zonei-tampon ; coloana RCS, care indică, în zecimal, lungimea zonei-articol ; coloana RCL, care indică numărul registrului utilizat pentru memorarea adresei articolului, în cazul în care RAM=MOV ; coloana ATDF, care indică, în hexazecimal, adresa de implantare, în cadrul modulului date, a tabelei de descriere a fișierului ; coloana ARCA, care indică, în hexazecimal, adresa de implantare a zonei-articol în cadrul modulului de date.

*Situația tipărită de compilatorul* FORTRAN grupează cinci tipuri de informații referitoare la :

– adresele de implantare (în hexazecimal) în cadrul modulului de program a primelor instrucțiuni în limbaj-mașină ale secvențelor ce corespund instrucțiunilor FORTRAN etichetate (în coloana ETIQ sînt indicate etichetele, iar în coloana ADR, adresele) ;

– numele, tipul, clasa, adresele de implantare în cadrul modulelor și lungimea simbolurilor definite în program (numele simbolului este indicat în coloana SYMBOLE ; tipul este indicat în coloana TYPE prin unul din caracterele I, R, D, C, L sau a căror semnificație este : variabilă de tip întreg, real simplă precizie, real dublă precizie, complex, logic, sau că simbolul reprezintă un parametru formal ; clasa este indicată, în coloana CLAS, prin caracterele S, T sau SP a căror semnificație este : variabilă obișnuită, tablou sau subprogram ; coloana ADRESSE poate conține fie adresa de implantare, în hexazecimal, a variabilei în cadrul modulului, adresă urmată de unul din caracterele P, D sau C care indică apartenența variabilei la modulul de program, de date sau de date comune, fie simbolul DUMMY, dacă variabila reprezintă un parametru formal ; lungimea zonei de memorie rezervată variabilei este indicată, în hexazecimal, în coloana LONG) ;

– lista variabilelor locale (care conține, pentru fiecare variabilă ce nu aparține unui bloc de tip „common", numele și adresa de implantare în cadrul modulului) ;

– lista variabilelor ce aparțin blocurilor de tip „common" (pentru fiecare bloc de tip „common" este tipărită o astfel de listă, care indică : numele blocului, numele variabilelor și adresele lor de implantare în cadrul modulului de date comune ce corespunde blocului) ;

– lista subprogramelor utilizator și a celor din BSS pentru care compilatorul a generat secvențe de apel.



Scanned with OKEN Scanner

*Erori semnalate la compilare (asamblare).* Erorile care se produc la scrierea (codificarea) programelor se datoresc :

– sintaxei incorecte a construcțiilor limbajului (de exemplu, instrucțiunea MULTIPLY A BY 200, în limbajul COBOL) ;

– ortografierii greșite a textului (de exemplu, utilizarea semnului „liniuță" în locul semnului „subliniere", a literei I în locul cifrei 1, a cifrei 0 în locul literei O, a punctului în locul virgulei, a semnului ; în locul semnului :, utilizarea incorectă sau omiterea delimitatorilor, nerespectarea ortografiei simbolurilor, omiterea unor clauze etc.) ;

– incompatibilității între construcțiile (instrucțiuni, clauze, operanzi etc.) limbajului (de exemplu, rubrica SELECT FISIER ASSIGN TO AMT ORGANIZATION DIRECT, în limbajul COBOL).

Erorile de primul tip sînt întotdeauna semnalate de programele traducătoare, pe cînd celelalte sînt sau nu semnalate (deoarece, în multe cazuri, sintaxa construcțiilor poate fi corectă).

Erorile detectate și semnalate de programele traducătoare le sînt asociate pentru nivele de gravitate, notate 1, 2, 3 și 4 :

– *nivel 1* : presupunere de eroare, avertisment, sintaxă incorectă (de exemplu : lipsa unei paranteze în cadrul unei expresii aritmetice, trunchierea valorii unei date, utilizarea fără sens a adresării cu indexare); programul traducător corijează sau nu eroarea ;

– *nivel 2* : erori puțin grave (de exemplu : variabilă inițializată de mai multe ori, lipsa unui apostrof ce trebuie să delimiteze un literal numeric, simbol dublu definit) ; programul traducător corijează eroarea conform unei anumite opțiuni ;

– *nivel 3* : erori grave (de exemplu : lipsa indicelui la adresarea unui element de tablou, sintaxă eronată a unei instrucțiuni) ; programul traducător nu generează codul obiect corespunzător construcției eronate ;

– *nivel 4* : erori foarte grave (de exemplu : dimensiunea unui bloc de tip „common" este mai mare decît 64 K baiți, depășirea capacității tabelelor programului traducător) ; este oprită traducerea, în continuare, a textului sursă.

Erorile cu nivel de gravitate 1 și unele erori de gravitate 2 (pentru care programul traducător alege opțiunea justă de corectare) nu afectează editarea legăturilor și execuția programului. În schimb, erorile cu nivel de gravitate 2 și 3 conduc la rezultate imprevizibile în faza de execuție a programului, iar cele cu nivel de gravitate 4, interzic editarea legăturilor. În mod obișnuit (adică, în absența parametrului ER : *n* în cartela de comandă COMPILE) nivelul maxim de gravitate a erorilor admis pentru trecerea la faza de editare a legăturilor este 2. Totuși, prin parametrul ER : *n* dacă $n=1$, se poate interzice editarea legăturilor și execuția programelor ce au erori cu nivel de gravitate 1, sau, dacă $n=3$,



Scanned with OKEN Scanner

4, se poate autoriza efectuarea acestor faze și în cazul programelor ce au erori cu nivel de gravitate 2 și 3.

Erorile detectate de programele traducătoare sînt imprimate la sfîrșitul (normal sau anormal) compilării pe suportul fișierului de ieșire al sistemului. În mod obișnuit, erorile sînt tipărite în ordinea numerelor liniilor sursă în care au fost detectate, pentru fiecare eroare indicîndu-se numărul liniei sursă și, eventual, coloana din cadrul liniei, codul erorii, nivelul de gravitate a erorii, eventual construcția eronată, mesajul care interpretează eroarea. În general, mesajele care interpretează erorile sînt destul de explicite, lucru ce permite identificarea și corectarea lor de către programator.

11.3.2. **Editarea legăturilor.** În această fază, modulele obiect sînt grupate pe tipuri, în unități independente – segmente – care alcătuiesc un *program executabil* sau *program imagine memorie translatabilă* (numit astfel, deoarece adresele datelor și instrucțiunilor au semnificație numai în cadrul programului, nefiind încă asociate zonelor reale ale memoriei în care se vor găsi segmentele programului în momentul execuției). Modulele obiect pot proveni din : compilări realizate în aceeași lucrare în care se realizează și editarea legăturilor, biblioteci de tip BT ale utilizatorului sau biblioteci de subprograme standard ale sistemului, fișierul de intrare al sistemului (cazul modulelor obiect perforate pe cartele). Rezultatul editării legăturilor este înregistrat în fișierul LOAD GO, de unde, în funcție de indicațiile utilizatorului, poate fi încărcat în memorie și executat, poate fi catalogat într-o bibliotecă de tip IMT sau perforat pe cartele, în vederea unor execuții ulterioare.

Editarea legăturilor se realizează în patru faze :

1) *Faza de inițializare*, în care sînt analizate opțiunile specificate în cartela LINK.

2) *Faza rezolvării referirilor externe*, în care se stabilesc legăturile între modulele ce alcătuiesc segmentele, cît și legăturile între segmentele programului. Tot în această fază, este analizată lista punctelor de intrare în modelele din BSS sau din biblioteci de subprograme ale utilizatorului, în scopul includerii acestora în segmentele programului.

3. *Faza generării segmentelor*, în care, pe baza textelor modulelor obiect din fișierul FILEDIT, sînt generate textele segmentelor. Corespunzător tipurilor de module vor fi generate : segmente de program, segmente de date și segmente de date comune. În general, un anumit tip de segment grupează unul sau mai multe module de același tip. Structura diferitelor tipuri de segmente este funcție de limbajul în care este scris programul (vezi cap. 10).

Ansamblul segmentelor generate reprezintă un program executabil pentru care nu sînt rezolvate referirile externe ce corespund modulelor



Scanned with OKEN Scanner

de acces SGF. Structura acestui program, care poate fi liniară sau arborescentă, este indicată de utilizator prin cartela TREE (în cazul în care cartela TREE nu este precizată, editorul de legături înlănțuie segmentele liniar).

La sfîrșitul fazei a treia, editorul de legături tipărește o situație, în care, pentru fiecare segment, se indică : numele, numărul de ordine în cadrul programului, adresa relativă de implantare în cadrul programului, numele modulelor care îl compun și adresele de implantare a acestora în în cadrul segmentului.

4) *Faza includerii modulelor de acces* SGF, în care, pe baza informațiilor din descrierea fișierelor (tipul unității periferice, organizarea fișierului, modul de prelucrare și modul de gestiune a zonelor-tampon), sînt selectate (din BSS) și adăugate programului modulele de acces SGF care vor realiza operațiile de intrare/ieșire (dacă în program sînt definite mai multe fișiere cu aceleși caracteristici, editorul de legături va selecta și încorpora în program un singur modul de acces). Totodată, în această fază, editorul de legături realizează rezervarea, inițializarea și inserarea în program a blocurilor de gestiune a directivelor (care conțin programele de canal și blocurile de comandă ce le sînt asociate) pentru fișierele definite în program și a modului de legătură SGF (care reprezintă o zonă de lucru a SGF-ului prin intermediul căreia se realizează funcțiile SGF : funcția de acces global la fișiere, funcțiile ASSIGN, LABEL, FILE etc.). De asemenea, informațiile ASSIGN, LABEL și FILE referitoare la fișierele definite în program sînt înregistrate într-o zonă numită ZONCOM.

În vederea încărcării corecte în memorie a programului, editorul de legături stabilește : numele segmentelor și adresele de implantare a acestora în cadrul programului (numele sînt grupate într-o zonă numită NAMSEG, iar adresele într-o zonă numită TABSEG) : adresa primei instrucțiuni ce va fi executată (adresa de lansare) ; adresa de implantare a modulului de legătură SGF ; lungimea totală a programului ; lungimea celei mai mari ramuri ; lungimea zonei de memorie necesară execuției programului (aceste informații sînt generate într-o zonă numită PROGID). Programul executabil, căruia îi sînt asociate informațiile PROGID, ZONCOM, NAMSEG și TABSEG, este apoi înregistrat în fișierul LOAD GO. În final, editorul de legături tipărește o situație care indică : numele, numărul de ordine, adresa de implantare și lungimea fiecărui segment ; numărul erorilor detectate ; adresa de lansare ; lungimea celei mai mari ramuri a programului ; lungimea totală a programului ; nivelul maxim de gravitate al erorilor semnalate.

*Erori semnalate la editarea legăturilor.* Cauzele care generează aceste erori sînt multiple : sintaxa sau conținutul cartelelor adresate editorului de legături sînt eronate, grupare greșită a molulelor în cadrul segmentelor sau a segmentelor în cadrul programelor, definire incorectă a rela-



Scanned with OKEN Scanner

ţiilor între modulele ce compun segmentele sau a relaţiilor între segmentele programului, informaţiile furnizate prin cartelele ASSIGN, FILE, LABEL referitoare la fişierele definite în program sînt incomplete sau incompatibile cu cele din descrierea fişierelor etc.

Erorile detectate la editarea legăturilor, grupate pe patru nivele de gravitate, sînt tipărite pe suportul fişierului de ieşire al sistemului. Fiecărei erori îi este asociat un cod (care reprezintă un număr format din patru cifre, prima cifră indicînd nivelul de gravitate) şi un mesaj care interpretează eroarea. În cazul în care nivelul de gravitate este 4 sau este egal sau mai mare decît cel precizat în cartela LINK (prin argumentul ER : $n$, $n=2, 3, 4$), generarea programului executabil este oprită.

În tabelul 11.1 sînt prezentate principalele erori semnalate la editarea legăturilor.

*Tabelul 11.1*

**Erori semnalate la editarea legăturilor**

| Număr eroare | Semnificaţie |
|---|---|
| 1010 | Sintaxa cartelei de comandă XTRACT este incorectă |
| 1050 | Adresa de lansare a programului nu a fost definită |
| 1060 | Două definiri externe cu acelaşi nume figurează în două segmente ce vor fi prezente simultan în memorie. |
| 1070 | Identificatorul de exploatare citat în cartela de comandă FILE, ASSIGN, LABEL, FLSET nu corespunde nici unuia dintre identificatorii de exploatare indicaţi în descrierea fişierelor definite în program |
| 1101 | Sintaxa cartelei de comandă „continuare" este incorectă |
| 2013 | Două puncte de intrare sînt identice |
| 2014 | Două puncte de intrare sînt identice (unul fiind stabilit de compilatorul FORTRAN) |
| 2060 | Prima cartelă de comandă SEG a programului este precedată de modulele obiect sau de alte cartele de comandă adresate editorului de legături |
| 2061 | Două cartele de comandă SEG definesc acelaşi segment |
| 2062 | În cartela de comandă SEG figurează un argument incorect |
| 2070 | Două definiri externe identice apar în acelaşi segment |
| 2080 | Două module obiect cu acelaşi nume apar în două segmente ce vor fi prezente în memorie simultan |
| 2094 | Referirea externă nu poate fi rezolvată (definirea externă neexistînd, sau există într-un segment care nu este citat în directiva USING) |
| 2100 | Adresa de lansare a programului reprezintă o referire externă nerezolvată |
| 2101 | Adresa de lansare a programului reprezintă o referire externă ce nu este definită în segmentul rădăcină |
| 2110 | Sintaxa cartelei de comandă FILE este incorectă |
| 2111 | Sintaxa cartelei de comandă LABEL este incorectă |
| 2112 | Sintaxa cartelei de comandă ASSIGN este incorectă |
| 2113 | Sintaxa cartelei de comandă FLSET este incorectă |
| 2118 | Selectarea modulului de acces SGF nu se poate realiza (descrierea fişierului este incompletă) |
| 2119 | Macroinstrucţiunile (instrucţiunile) de acces la înregistrările fişierului sînt incompatibile cu tipul unităţii periferice |



Scanned with OKEN Scanner

*Tabelul 11.1* (continuare)

| Număr eroare | Semnificație |
|---|---|
| 3000 | Sintaxa cartelei de comandă TREE este incorectă |
| 3001 | În cartela de comandă TREE parantezele nu sînt în număr par |
| 3002 | În cartela de comandă TREE descrierea structurii arborescente a programului este incorectă |
| 3004 | Un segment citat în cartela de comandă TREE nu a fost definit în cartela de comandă SEG |
| 3005 | În cartela de comandă TREE un segment este citat de două ori |
| 3006 | În cartela de comandă TREE nu este citat un segment definit în cartela de comandă SEG |
| 3030 | Modulul de acces SGF nu poate fi încorporat programului deoarece nu se găsește în biblioteca sistemului |
| 4010 | Lungimea segmentului depășește 64 K baiți |
| 4040 | Depășirea tabelei de definiri externe |
| 4042 | Depășirea tabelei de puncte de intrare în modul (subprogram) |
| 4048 | Spațiu insuficient pentru continuarea editării legăturilor |
| 4050 | Grupa (familia) subprogramelor X% (module SGF) nu se află în biblioteca sistemului. |

11.3.3. **Încărcarea programelor în memorie.** Pentru a fi executate, programele trebuie să fie înregistrate în memoria centrală, operație realizată prin intermediul programului de încărcare al sistemului de operare (încărcător). Programele acceptate de încărcător se pot afla în fișierul LOAD GO, în BUS, într-o bibliotecă de tip IMT a utilizatorului sau în fișierul de lucrări (cazul programelor IMT perforate pe cartele). În timpul operației de încărcare se realizează :

– stabilirea adresei zonei-memorie de la care va începe încărcarea programului ;

– translatarea finală a adreselor de tip RA ;

– inițializarea zonei ZONCOM, în cazul în care cartelele ASSIGN, LABEL și FILE referitoare la fișierele definite în program au fost plasate în cadrul lucrării după cartela LINK ;

– definitivarea tabelei segmentelor ;

– înregistrarea în memorie a segmentului de program și, eventual, a segmentului de date și a segmentelor de date comune care îi sînt asociate în cazul programelor nesegmentate sau a segmentului rădăcină și, eventual, a segmentelor de date și de date comune care sînt asociate în cazul programelor segmentate ;

– încărcarea în registrul de bază (care este întotdeauna registrul R14, în cazul programelor COBOL și FORTRAN) a adresei de la care a fost înregistrat programul în memorie ; totodată în registrul R13 este încărcată adresa segmentului de date, în cazul programelor COBOL,



Scanned with OKEN Scanner

sau adresa unei zone de lucru comună tuturor modulelor FORTRAN apelate din BSS, în cazul programelor FORTRAN ; în registrele R9–R12 sînt încărcate adresele segmentelor de date comune, în cazul programelor FORTRAN segmentate prin utilizarea directivei * SEGMENT, sau adresele segmentului de date şi a segmentelor de date comune, în cazul programelor FORTRAN segmentate prin utilizarea directivei * SEGDATA) ;

– înregistrarea în zona de memorie alocată programului a tabelei segmentelor şi încărcarea în registrul R15 a adresei acestei tabele ;

– încărcarea în registrul de stare program a adresei de start (adresa de lansare) a programului şi lansarea execuţiei acesteia.

Zona de memorie alocată unui program (numită şi partiţie sau regiune) este structurată de încărcător în patru subzone – *zona de informaţii generale*, *tabela segmentelor*, *zona de prelucrare* şi *zona de opţiuni* – în care sînt înregistrate informaţiile ce alcătuiesc programul propriu-zis, precum şi anumite informaţii care îi sînt asociate de sistem, în vederea execuţiei.

*Zona de informaţii generale* (ZIG) are dimensiune şi structură fixe, care sînt stabilite la generarea sistemului de operare. Informaţiile grupate în această zonă sînt înregistrate la iniţializarea sistemului, în momentul începerii execuţiei unei lucrări (JOB), în timpul încărcării şi execuţiei programelor ce compun lucrarea, fiind utilizate atît de către sistem cît şi de către utilizator (prin intermediul limbajului de asamblare). Principalele informaţii înregistrate în această zonă sînt prezentate în tabelul 11.2.

*Tabelul 11.2*

**Zona de informaţii generale**

| Adresa relativă (în baiţi) | Semnificaţie |
|---|---|
| 0—4 | Adresa modulului de legătură SGF |
| 4—7 | Adresa rezumatului SGF |
| 8—15 | Numele lucrării |
| 18 | Codul de retur al fazei precedente |
| 20—30 | Data zilei |
| 32—35 | Adresa ultimei locaţii a partiţiei |
| 36—39 | Adresa primei locaţii a zonei de opţiuni |
| 40—43 | Adresa ultimului bait al ultimului segment încărcat |
| 45—47 | Adresa ultimului bait al celei mai lungi ramuri |
| 51 | Contor lucrări |

*Tabela segmentelor* reprezintă o zonă de memorie de lungime $n+1$ cuvinte-memorie, unde $n$ este numărul segmentelor ce alcătuiesc programul. Primul cuvînt reprezintă adresa de început a zonei de informaţii generale, iar următoarele cuvinte, adresele de la care vor fi încărcate în memorie segmentele programului. Adresele de implantare a segmen-



Scanned with OKEN Scanner

telor sînt înregistrate în această tabelă conform ordinii atribuite segmentelor la editarea legăturilor. Prima cifră binară a fiecărui cuvînt ce reprezintă adresa de implantare a unui segment indică starea acestuia şi anume : 0 – segmentul este încărcat în memorie ; 1 – segmentul nu este încărcat în memorie.

În *zona de prelucrare* sînt încărcate segmentele ce alcătuiesc programul. În cazul programelor nesegmentate toate segmentele ce le compun sînt înregistrate în zona de prelucrare în faza de încărcare. În cazul programelor segmentate cu structură liniară sau arborescentă, în faza de încărcare este înregistrată, în zona de prelucrare, rădăcina programului (care conţine segmentul de program şi, eventual, segmentele de date şi de date comune care îi sînt asociate), celelalte segmente fiind încărcate în timpul execuţiei programului.

Textul fiecărui segment de program este precedat de *tabela punctelor de intrare*, care conţine cîte un cuvînt-memorie pentru fiecare punct de intrare în segment ; acest cuvînt reprezintă adresa de tip BA a punctului de intrare. Adresa zonei de prelucrare coincide cu adresa de implantare a primului segment de program.

*În zona de opţiuni* sînt înregistrate, în momentul încărcării programului în memorie, informaţiile specificate în cartela de comandă OPTION. Accesul la aceste informaţii, care se poate realiza numai prin intermediul limbajului de asamblare, permite cumunicarea între fazele de execuţie ale lucrării ce conţine programul (aceasta deoarece informaţiile înregistrate în zona de opţiuni nu sînt distruse decît în momentul încărcării unui alt program pentru care se specifică cartela OPTION). Dimensiunea zonei de opţiuni este fixă, fiind stabilită la generarea sistemului de operare, iar adresa ultimei locaţii a acestei zone este înregistrată în zona de informaţii generale.

*Erori semnalate la încărcare.* Aceste erori (vezi tabelul 11.3) sînt tipărite pe suportul fişierului de ieşire al sistemului, imediat după imaginea cartelei prin care s-a cerut încărcarea programului în memorie (RUN, SYSRUN, FETCH, COMPILE).

*Tabelul 11.3*

**Erori semnalate la încărcarea programelor**

| Mesaj | Semnificaţie |
|---|---|
| PROGRAMME NON TROUVE DANS LA BIBLIOTHEQUE UTILISATEUR STANDARD | Programul, a cărui încărcare este cerută, nu se găseşte în Biblioteca Utilizator Standard |
| ERREUR DANS LAN- | |



Scanned with OKEN Scanner

*Tabelul 11.3* (continuare)

| Mesaj | Semnificație |
|---|---|
| CEMENT DE PRO-GRAMME D'APRES POINT DE REPRISE | |
| IMPOSSM | Dimensiunea zonei de memorie asociată programului este insuficientă pentru încărcarea și execuția acestuia |
| IMPOSSD | Una (sau mai multe) dintre unitățile periferice asociate fișierelor definite în program nu poate fi afectată |
| IMPOSSF | Zona FILECOM este saturată |
| IMPOSSL | Unul sau mai multe module de acces SGF nu au fost încorporate (la editarea legăturilor) în program |
| IMPOSSC | Informațiile furnizate prin una (sau mai multe) dintre cartelele de comandă asociate fișierelor definite în program sînt eronate |
| IMPOSSN | Programul a cărui încărcare este cerută nu se găsește în Biblioteca Utilizator Standard |

## 11.4. ERORI SEMNALATE ÎN FAZA DE EXECUȚIE A PROGRAMELOR

Execuția unui program reprezintă o succesiune de execuții de instrucțiuni, succesiune ce depinde de program și de datele de prelucrare. Execuția unei instrucțiuni se realizează în trei etape : extragerea instrucțiunii din memorie, calculul adresei Y și efectuarea de către unitatea centrală a operației indicate de instrucțiune. În cadrul fiecărei etape unitatea centrală efectuează anumite operații elementare. Operațiile elementare efectuate în primele două etape sînt, în general, comune tuturor instrucțiunilor, pe cînd operațiile elementare efectuate în ultima etapă, sînt specifice fiecărei instrucțiuni. Execuția unui program începe cu execuția instrucțiunii a cărei adresă a fost înregistrată, în momentul încărcării, în registrul de stare program ; această instrucțiune se numește *instrucțiune de start*, iar adresa sa, *adresa de start* (de lansare).

Pe parcursul execuției unui program, unele instrucțiuni sînt executate de mai multe ori, altele o singură dată sau deloc. De asemenea, în cazul programelor segmentate, execuția presupune și încărcarea în memorie a segmentelor ce compun diferitele ramuri ale programului.

Erorile care apar la execuția programului pot fi : *erori de programare* (majoritatea), *erori hardware*, *erori software* și *erori de operare*.

În general, acestea sînt detectate de diferite componente ale sistemului de operare și tipărite la mașina de scris cît și pe suportul fișierului de ieșire al sistemului, semnalîndu-se astfel sfîrșitul anormal al fazei de execuție a programului.



Scanned with OKEN Scanner

În vederea identificării cauzelor care le-au generat, tipărirea mesajelor care interpretează aceste erori este urmată de tipărirea, pe suportul fișierului de ieșire al sistemului, a conținutului zonei de memorie în care programul este executat.

În continuare sînt prezentate principalele erori semnalate de monitor, sistemul de gestiune al fișierelor și de subprogramele din BSS, apelate de programele traducătoare.

*Cauzele care generează erorile semnalate de monitor* pot fi :

1) *Apariția unei situații deosebite interne programului.*

Formatul mesajului :

ERREUR PROGRAMME 0000*xxxx*

unde *xxxx* reprezintă codul erorii (vezi tabelul 11.4)

*Tabelul 11.4*

**Erori de tip** ERREUR PROGRAMME

| Cod eroare (*xxxx*) | Semnificația |
|---|---|
| 0101 | Depășire de calcul în virgulă fixă (depășire binară) |
| 0102 | Depășire de calcul zecimal (depășire zecimală) |
| 0104 | Depășire inferioară în virgulă mobilă |
| 0108 | Depășire superioară în virgulă mobilă |
| 0201 | Instrucțiune privilegiată executată într-un program utilizator sub controlul sistemului de operare |
| 0202 | Adresarea unei locații de memorie inexistentă (unitatea de memorie nu conține blocul sau modulul corespunzător locației |
| 0204 | Adresarea unei locații de memorie protejată |
| 0208 | Depășirea timpului maxim admis pentru execuția instrucțiunii (de exemplu, numărul de cicluri efectuate pentru calculul adresei, în cazul adresării indirecte, a depășit valoarea 5) |
| 0400 | Codul operației indicat în instrucțiune nu este recunoscut de unitatea centrală (cod operație inexistentă) |
| 0800 | Excepție software |

2) *Programarea eronată a unei operații de intrare/ieșire sau apariția unui incident hardware în cursul unei operații de intrare/ieșire.*

Formatul mesajului :

*np* ERREUR E/S *nnaaaaaa*

unde *np* reprezintă numărul partiției, *nn* codul erorii, iar *aaaaaa* o adresă (vezi tabelul 11.5).



Scanned with OKEN Scanner

Erori de tip ERREUR E/S

*Tabelul 11.5*

| Cod eroare (*nn*) | Adresa (*aaaaaa*) | Semnificație |
|---|---|---|
| 0 | 1 | 2 |
| | | **Erori de programare** |
| 01 | Adresa instrucțiunii care urmează instrucțiunii de lansare a cererii pentru operația de intrare/ieșire (EXUP) | Adresa blocului de comandă nu este cuprinsă între limitele partiției (regiunii) în care este executat programul |
| 02 | Adresa blocului de comandă | Adresa programului de canal nu este cuprinsă între limitele partiției (regiunii) în care este executat programul |
| 03 | Adresa blocului de comandă | Unitatea periferică nu este afectată |
| 04 | Adresa blocului de comandă | Adresa zonei-tampon nu este cuprinsă între limitele partiției (regiunii) în care este executat programul (cazul fișierului de intrare al sistemului) |
| 05 | Adresa blocului de comandă | Eroare de programare a unei operații de intrare/ieșire pe disc |
| 06 | Adresa blocului de comandă | Detectarea sfîrșitului de bandă, în cazul fișierelor de ieșire sau de perforare ale sistemului care au ca suport banda magnetică |
| 07 | Adresa blocului de comandă | Numărul logic asociat unității periferice este incorect |
| 08 | Adresa instrucțiunii care urmează instrucțiunii de lansare a cererii pentru operația de intrare/ieșire (EXUP) | Macroinstrucțiunea RST este incorectă |
| 09 | Adresa blocului de comandă | Prima comandă a programului de canal ce descrie o operație de intrare/ieșire pe disc nu este o comandă de poziționare |



Scanned with OKEN Scanner

*Tabelul 11.5* (continuare)

| 0 | 1 | 2 |
|---|---|---|
| 10 | Adresa blocului de comandă | Citirea cartelei de comandă JOB de către programul utilizator |
| 11 | Adresa blocului de comandă | Încercare de acces la fișier fără deschidere prealabilă |
| 12 | Adresa blocului de comandă | A fost depășit numărul de linii/cartele perforate (cazul fișierelor de ieșire și de perforare ale sistemului) |
| 13 | Adresa blocului de comandă | Zona alocată fișierului de ieșire sau de perforare al sistemului (care are ca suport discul) este insuficientă |
| 14 | Adresa blocului de comandă | Numărul de octeți incorect (cazul fișierelor de ieșire și de perforare ale sistemului) |
| | | **Erori hardware** |
| 20 | Adresa blocului de comandă | Incident ireparabil pe disc |
| 21 | Adresa fizică a unității periferice | Încercare de scriere pe un volum disc protejat la scriere |
| 22 | Adresa blocului de comandă | Cerere de efectuare a operației de intrare/ieșire în cazul unei unități periferice off-line |
| 23 | Adresa blocului de comandă | Eroare de ritm pe disc |
| 24 | Adresa blocului de comandă | Trecerea în mod „manual" a unității de bandă magnetică, în cursul efectuării operației de intrare/ieșire |
| 25 | Adresa blocului de comandă | Incident ireparabil pe disc de tip DIMAS |
| 26 | Adresa fizică a unității periferice | Absența întreruperii, semnalată de watchdog (cazul unităților periferice electromagnetice) |



Scanned with OKEN Scanner

3) *Operatorul cere sfîrșitul execuției programului, datorită apariției unui incident hardware la o unitate periferică.*

Formatul mesajului :

DECISION OPERATEUR *nn aaaaaa*

unde *nn* reprezintă codul erorii, iar *aaaaaa* adresa blocului de comandă sau adresa fizică a unității periferice.

**Observație.** Apariția unui incident hardware la o unitate periferică este semnalată de monitor, la mașina de scris prin două tipuri de mesaje :

— DEV *aaa* MANUEL MODE

unde *aaa* reprezintă adresa fizică a unității periferice ;

— *np* ERR *nn mm . . . m*

unde *np* reprezintă numărul partiției, *nn* codul erorii, iar *mm . . . m* un mesaj de interpretare a erorii, adresa unității periferice, adresa blocului de comandă, răspunsul furnizat prin execuția instrucțiunii AIO sau adresa programului de canal (vezi tabelul 11.6).

În cazul în care incidentul nu poate fi înlăturat, iar operatorul cere sfîrșitul execuției programului, pe suportul fișierului de ieșire al sistemului este tipărit mesajul

DECISION OPERATEUR *nn aaaaaa*

(*nn* are semnificația prezentată în tabelul 11.6)

*Tabelul 11.6*

**Mesaje tipărite la mașina de scris în cazul erorilor de tip** ERREUR E/S

| Număr eroare (*nn*) | Informația furnizată (*xx . . . x*) | Semnificație |
|---|---|---|
| 0 | 1 | 2 |
| 00 | Text de interpretare a erorii și adresa fizică a unității periferice | Unitate periferică electromagnetică neoperațională sau în mod „manual" ; absența hîrtiei de imprimat ; avansul continuu al hîrtiei de imprimat |
| 13 | Adresa unității periferice și a blocului de comandă | Lungime incorectă |
| 14 | Adresa unității periferice și a blocului de comandă | Eroare de ritm (cazul unităților periferice electromagnetice) |
| 15 | Adresa unității periferice și a blocului de comandă | Încercare de scriere pe un volum bandă protejat la scriere |
| 16 | Adresa unității periferice și răspunsul furnizat în urma execuției instrucțiunii AIO | Eroare de paritate (în cazul imprimantei) |
| 17 | Adresa unității periferice și a blocului de comandă | Incident ireparabil la citire de pe bandă magnetică sau la perforarea pe cartele |
| 19 | Adresa unității periferice și a blocului de comandă | Incident ireparabil la scriere pe bandă magnetică |
| 21 | Adresa unității periferice și a blocului de comandă | Încercare de scriere pe un volum disc protejat la scriere |



Scanned with OKEN Scanner

*Tabelul 11.6* (continuare)

| 6 | 1 | 2 |
|---|---|---|
| 22 | Adresa unității periferice și a blocului de comandă | Adresa fizică a unității periferice nu a fost recunoscută |
| 24 | Adresa unității periferice și răspunsul furnizat în urma execuției instrucțiunii AIO | Codificarea caracterelor pe cartelă nu corespunde codificării Hollerith |
| 27 | Adresa unității periferice | Unitate periferică neoperațională |
| 28 | Adresa unității periferice | Citirea pe o unitate de bandă magnetică cu densitatea de înregistrare de 1 600 bpi a unei benzi înregistrate cu densitatea de 800 bpi |
| 29 | Adresa unității periferice și a blocului de comandă | Detectarea unui bloc „parazit" |

4) *Operatorul cere, de la panoul de comandă, sfîrșitul execuției programului.*

Formatul mesajului :

INTERVENTION OPERATEUR

5) *Este apelată o funcție a supervizorului* (CSV), *pentru care însă parametri specificați sînt incompleți sau eronați.*

Formatul mesajului :

DEMANDE INSATISFAITE *aaaaaaaa*

unde *aaaaaaaa* reprezintă adresa instrucțiunii ce urmează instrucțiunii de apel al funcției.

6) *Apariția unui incident software.*

Formatul mesajului :

ERREUR SYSTEME

*Cauzele erorilor* semnalate de SGF sînt foarte variate : descrieri eronate a fișierelor, incompatibilități între parametrii de descriere, montări greșite a volumelor, utilizări greșite a instrucțiunilor sau macroinstrucțiunilor SGF, dimensionări greșite a fișierelor, distrugeri, prin erori de programare, a unor tabele sau informații SGF, apariția unor incidente hardware în cursul prelucrării fișierelor etc.

Erorile detectate de către SGF pot face sau nu obiectul unei intervenții a operatorului. Astfel, în cazul erorilor ce pot fi corijate de operatori (de exemplu : montări greșite a volumelor) acestea sînt semnalate la mașina de scris. În cazul unor erori grave (de exemplu, erori de programare), pentru corijarea cărora operatorul nu poate interveni, acestea



Scanned with OKEN Scanner

sînt semnalate și pe suportul fișierului de ieșire al sistemului (indicînd sfîrșitul anormal al fazei de execuție a programului), conform următorului format:

*idex* EGF *cc nn mm ... m*

unde *idex* reprezintă identificatorul de exploatare al fișierului la prelucrarea căruia s-a semnalat eroarea, *cc* reprezintă codul segmentului sau modulului de acces SGF care a detectat eroarea, *nn* reprezintă codul erorii, iar *mm ... m* reprezintă mesajul care interpretează eroarea (vezi tabelul 11.7).

*Tabelul 11.7*

**Erori semnalate de SGF**

| Cod eroare (*nn*) | Mesaj (*mm .. m*) | Semnificație |
|---|---|---|
| 0 | 1 | 2 |
| 01 | OUV.F.ACTIF | Fișierul a fost deja deschis |
| 04 | DVT ABSENT | Tipul unității periferice (DVT) asociată fișierului nu este precizat în TDF |
| 05 | PRM ABSENT | Tipul operației (modul de prelucrare) la care va fi supus fișierul nu este precizat în TDF |
| 06 | UPD INTRED | Operația de actualizare interzisă pentru un fișier pe bandă magnetică |
| 07 | DVT : TDF≠ASS | Tipul unității periferice definit în cartela ASSIGN diferă de cel indicat în descrierea fișierului |
| 09 | POSITION MT | Poziționare incorectă a dispozitivului de scriere/citire în cazul adăugării de înregistrări la sfîrșitul fișierului sau în cazul scrierii/citirii multifișier pe bandă magnetică |
| OA | MT. PROTEGEE | Adăugarea de înregistrări la sfîrșitul unui monofișier sau în cadrul unui multifișier pe bandă nu se poate realiza deoarece volumul este protejat la scriere |
| OB | MT. PROTEGEE | Crearea fișierului pe bandă magnetică nu se poate realiza deoarece volumul este protejat la scriere |
| 11 | EOF1 ABSENT | Etichetele de sfîrșit de fișier nu corespund cu cele ale unui fișier standard pe bandă sau poziționarea dispozitivului de scriere/citire pentru adăugarea de înregistrări fișierului (sau pentru citirea înapoi) este incorectă |
| 14 | FM MAL PLACE | Etichetele de identificare a fișierului nu corespund cu cele ale unui fișier standard pe bandă |
| 16 | F. NON PERIME | Ștergerea fișierului nu poate fi realizată deoarece perioada de păstrare a acestuia nu a expirat (data curentă este mai mică decît data de expirare) |
| 17 | HDR1 ABSENT | Etichetele de început de fișier nu corespund cu cele ale unui fișier standard pe bandă. |
| 21 | FN INCONNU | Fișier absent pe volum (bandă magnetică) |
| 22 | GEN INCONNU | Generația fișierului (GN) precizată în cartela LABEL nu corespunde celei din eticheta HDR1 (cazul fișierelor pe bandă magnetică) |



Scanned with OKEN Scanner

*Tabelul 11.7* (continuare)

| 0 | 1 | 2 |
|---|---|---|
| 25 | VN INCONNU | Versiunea fișierului (VN) precizată în cartela LABEL nu corespunde celei din eticheta HDR1 (cazul fișierelor pe bandă magnetică) |
| 29 | F. NON TROUVE | Fișierul nu se află pe volum ; volumul este montat greșit sau informațiile din cartela LABEL sînt incompatibile cu cele din cartela ALLOC |
| 33 | BFS ERRONE | Dimensiunea zonei-tampon indicată în descrierea fișierului diferă de cea specificată la crearea fișierului |
| 35 | F. DEJA CREE | Fișierul nu poate fi creat deoarece pe volum există deja un fișier cu aceleași caracteristici (cazul fișierelor pe disc magnetic) |
| 39 | F. NON ECRIT | Fișierul nu poate fi citit deoarece nu a fost creat |
| 41 | E/S ERRONNE | Incident hardware ireparabil, la citirea etichetelor fișierului |
| 43 | HDR1 ABSENT | Fișierul nu poate fi citit deoarece : nu este primul pe volum (cazul multifișierelor pe bandă magnetică) ; etichetele de identificare nu corespund cu cele ale unui fișier standard pe bandă magnetică ; dispozitivul de scriere/citire este poziționat incorect |
| 44 | F. NON ACTIV | Fișierul nu poate fi închis deoarece nu a fost deschis anterior |
| 46 | CARTE. ERR | Din fișier a fost citită o cartelă cu punct în coloana 1, diferită de cartelele EOF sau CKP |
| 4A | MFF ERRONE | Modul de prelucrare precizat în descrierea fișierului necesită prezența contorului de actualizare |
| 51 | E/S ERRONE | Incident hardware ireparabil, la deschiderea sau închiderea fișierului |
| 53 | IDEX*ERR. | Identificatorul de exploatare al fișierului nu este compatibil cu unitatea periferică asociată unui fișier sistem |
| 61 | TOV SATUREE | Fișierului nu i se poate aloca zonă deoarece nu mai este loc pe volum |
| 62 | XU REMPLIE | Dimensiunea zonei de extensie alocată fișierului este insuficientă |
| 63 | DVN TDV≠ASS | Numărul unităților periferice afectate fișierului, specificat în TDF, diferă de cel indicat în cartela ASSIGN |
| 67 | SZ ABSENT | Argumentul SZ este absent (cazul fișierelor pe disc create pe volume încărcate „în paralel“) |
| 6E | CLOSE I IMP | Închiderea fișierului (de organizare secvențială-indexată) nu poate fi realizată, deoarece nu a fost creat |
| 71 | NO. APP. ABS | Unitatea periferică asociată fișierului nu este disponibilă |
| 78 | RCF ERRONE | Formatul articolelor, precizat în descrierea fișierului, diferă de cel indicat la crearea fișierului |
| 79 | KYS ERRONE | Dimensiunea cheii de identificare a articolelor, precizată în descrierea fișierului, diferă de cea indicată la crearea fișierului (cazul fișierelor secvențiale-indexate și selective) |



Scanned with OKEN Scanner

*Tabelul 11.7* (continuare)

| 0 | 1 | 2 |
|---|---|---|
| 84 | BFS INC. RCS. | Dimensiunea înregistrării logice este incompatibilă cu dimensiunea înregistrării fizice |
| 91 | BIN<2 | La crearea unui fișier secvențial-indexat numărul zonelor-tampon pentru tabelele de index trebuie să fie cuprins între 2 și 5 |
| 92 | DIM. CASE ER | Dimensiunea căsuței este incorectă (cazul fișierelor selective) |
| 93 | BIS<2 | Dimensiunea zonei-tampon pentru tabelele de index este mai mică decît 256 locații |
| 01 | F. NON OUVERT | Fișierul nu poate fi prelucrat deoarece nu este deschis |
| 02 | E/S ERRONEE | Incident hardware în cursul execuției unei operații de intrare-ieșire |
| 03 | CCB>BFN | Numărul atribuit blocului de gestiune a directivelor este mai mare decît numărul de zone-tampon afectate fișierului |
| 07 | ZP REMPLIE | În zona partajată asociată fișierului nu mai există nici o unitate de alocare liberă |
| 09 | GET/SLT ABS | Macroinstrucțiunea RPL sau DLT nu este precedată de macroinstrucțiunea GET sau SLT (rescrierea sau ștergerea unei înregistrări a fișierului nu este precedată de citirea acestuia) |
| 15 | OCT. INV. ABS. | Articolul nu poate fi șters deoarece nu i s-a prevăzut indicator de ștergere (cazul fișierelor pe disc) |
| 16 | DEPSMT ZONE | Încercare de scriere a fișierului într-o zonă care nu i-a fost alocată |
| 17 | BLOC ABSENT | Incident hardware în cursul execuției operației de citire a unei înregistrări fizice |
| 31 | ARTICLE ERR | Lungimea înregistrării fizice, a înregistrării logice sau a antetului înregistrării fizice este incorectă |
| 34 | BFS INSUFF. | Dimensiunea zonei-tampon nu poate conține un articol (cazul fișierelor selective) |

*Erorile detectate de subprogramele din* BSS, pentru care programele traducătoare generează secvențe de apel, sînt tipărite pe suportul fișierului de ieșire al sistemului sau la mașina de scris, conform următoarelor formate :

– ERREUR : *nnnnnn* (ADRESSE : *aaaaaa*) *mm . . .m*

în cazul subprogramelor apelate de compilatorul FORTRAN (*nnnnnn* reprezintă numele subprogramului care a detectat eroarea, *aaaaaa* reprezintă adresa instrucțiunii care a apelat subprogramul pentru execuție, iar *mm . . . m* reprezintă mesajul care interpretează eroarea) ;

– E/S SUR FICHIER NON OUVERT, OPEN SUR FICHIER OUVERT, CLOSE SUR FICHIER FERME (cazul fișierelor care utilizează unitățile standard ale sistemului) și READ SUR FICHIER NON



Scanned with OKEN Scanner

OUVERT.ABORT. WRITE SUR FICHIER NON OUVERT. ABORT (cazul celorlalte fișiere) în cazul subprogramelor apelate de compilatorul COBOL ;

— ERREUR : OPT.SRG (ADRESSE : *aaaaaa*) VALEUR D'INDICE INCORRECTE sau 01 EN *xxxxxx* DEBORDEMENT DE L'INDICE : *v* — în cazul inserării (prin opțiunea SRG în cartela COMPILE), în programele FORTRAN sau COBOL, a subprogramului de control al valorilor indicilor (*aaaaaa* reprezintă adresa instrucțiunii care a apelat programul, *xxxxxx* adresa indicelului, iar *v* valoare acestuia).

## 11.5. FACILITĂȚI DE TESTARE ȘI PUNERE LA PUNCT A PROGRAMELOR

Realizarea testării și punerii la punct a programelor impune ca programatorul să stabilească și să comunice sistemului de operare care aspecte ale funcționării programelor dorește să le examineze și care sînt informațiile ce îl pot ajuta la localizarea și stabilirea cauzelor erorilor. De asemenea, sistemul de operare trebuie să fie capabil să răspundă solicitărilor programatorului, iar mesajele informative sau de interpretare a erorilor pe care le furnizează să fie explicite și clare. În general, sistemele de operare moderne răspund acestor cerințe, avînd implementat dialogul necesar testării și punerii la punct a programelor. Trebuie remarcat însă faptul că, mesajele de interpretare a anumitor erori sînt dificil de înțeles (fiind compuse din coduri și simboluri a căror semnificație este greu de stabilit), iar uneori aceste mesaje lipsesc.

În cele ce urmează sînt prezentate facilitățile de testare și punere la punct oferite de componentele sistemelor de operare ale calculatoarelor FELIX C-256 și IRIS-50 (programele utilitare, programele traducătoare, editorul de legături și monitorul) precum și modul cum pot fi utilizate.

11.5.1. **Programe utilitare.** Două programe utilitare prezintă un interes deosebit pentru testarea și punerea la punct a programelor : generatorul de fișiere de test și programul de inventariere a fișierelor.

Generatorul de fișiere de test [134] permite simularea fișierelor (secvențiale, pe suport magnetic) definite în programele supuse testării. Situațiile care impun simularea fișierelor reale pot fi : inexistența acestora în momentul testării, volumul mare al datelor pe care le conțin (care face ineficientă testarea), datele ce le conțin nu sînt relevante pentru testare (nepermițînd testarea tuturor ramificațiilor logice ale algoritmului de prelucrare), posibilitatea alterării lor (cazul fișierelor supuse operației de actualizare) etc.



Scanned with OKEN Scanner

Programul de inventariere a fişierelor [133], permite imprimarea parţială sau totală a înregistrărilor logice sau fizice a fişierelor (de orice organizare), create sau actualizate prin programele supuse testării. Acest program poate fi utilizat atît în faza de testare cît şi în faza de punere la punct a programelor.

11.5.2. **Instrucţiuni şi rutine pentru testare şi punere la punct.** Inserarea în programe a unor instrucţiuni (sau secvenţe de instrucţiuni) care să realizeze testarea unor condiţii, controlul valorilor indicilor, marcarea trecerii controlului execuţiei prin diferite puncte ale programului, semnalarea (prin mesaje la imprimantă) unor situaţii anormale (de exemplu, valori incorecte ale datelor de intrare) etc., este posibilă în toate cele trei limbaje: ASSIRIS, COBOL, FORTRAN, utilizîndu-se instrucţiunile obişnuite ale acestora. Problema care se pune în acest caz este stabilirea judicioasă a locului în care sînt plasate în program aceste instrucţiuni, astfel încît, să se minimizeze volumul de informaţii tipărit la imprimantă.

Înlăturarea automată, după testare, a acestor instrucţiuni, este posibilă numai în cazul programelor FORTRAN. Execuţia acestor instrucţiuni numai în faza de testare este indicată compilatorului FORTRAN prin opţiunea CDC sau DBG în cartela COMPILE şi prin caracterele C/ în coloanele 1 şi 2 ale liniilor ce conţin instrucţiunile respective. Absenţa opţiunii CDC sau DBG are ca efect considerarea acestor instrucţiuni drept comentarii.

Inserarea în cadrul programelor a unor rutine din BSS pentru controlul valorilor indicilor reprezintă o facilitate disponibilă atît în limbajul COBOL cît şi în limbajul FORTRAN şi este comunicată compilatoarelor respective prin intermediul opţiunii SRG a cartelei COMPILE. În cursul execuţiei programului, dacă un indice ia o valoare ce nu aparţine intervalului de valori definit de dimensiunea (dimensiunile) tabloului adresat, un mesaj de eroare este tipărit la maşina de scris (vezi 11.4).

O altă facilitate disponibilă în limbajul FORTRAN este includerea în programe a unei rutine din BSS pentru verificarea, în momentul apelării subprogramelor, a concordanţei (ca număr şi tip) parametrilor actuali cu cei formali. Inserarea acestei rutine este indicată prin opţiunea PCK sau DBG, în cartela COMPILE. Neconcordanţa parametrilor este semnalată, în momentul execuţiei, printr-un mesaj imprimat pe suportul fişierului de ieşire al sistemului.

11.5.3. **Rutine de trasare.** Includerea în programe a unor rutine de trasare este posibilă numai în limbajul COBOL, prin intermediul instrucţiunii PRINT, al cărei format este :

```
7      12
+      PRINT [ { nume-dată-1 } [ , { nume-dată-2 } ] ... ]
             { literal-1   }       { literal-2   }
```

În absența opțiunii DBG în cartela COMPILE, liniile sursă ce conțin aceste instrucțiuni sînt considerate comentarii. În caz contrar, prelucrarea lor de către compilatorul COBOL constă în generarea unor secvențe de apel al unei rutine din BSS care, în momentul execuției programului, va tipări la imprimantă numele paragrafelor în care aceste instrucțiuni sînt situate și, dacă opțiunile *nume-dată-1/literal-1* ... sînt utilizate, valorile acestora. De asemenea, editorul de legături facilitează, prin intermediul cartelei TRCALL, inserarea unei rutine de trasare care să marcheze, în momentul execuției programului, transferul controlului (apelul) diferitelor module și/sau segmente ale programelor. Apelul unui modul sau segment este consemnat de această rutină prin tipărirea, pe suportul fișierului de ieșire al sistemului, a următoarelor informații : conținutul registrului de stare a programului, conținutul registrelor generale, numărul segmentului, numărul și adresa punctului de intrare în segment și, eventual, numele acestui punct de intrare.

Formatul cartelei TRCALL este următorul :

$$\text{TRCALL} \quad [nume1]\ [,\ nume2\ [,\ intreg]]\ \left[,\ \text{LN} : \left\{\begin{matrix}\text{ALL}\\ \text{SP}\\ \text{SEG}\end{matrix}\right\}\right]\left[,\ \text{CL} : \left\{\begin{matrix}\text{NO}\\ \text{YES}\end{matrix}\right\}\right]$$

unde : *nume1* și *nume2* reprezintă numele punctelor de intrare, între care rutina de trasare va semnala apelul modulelor și/sau segmentelor (în absența acestor argumente, rutina de trasare va lua în considerare apelurile din întregul program) ;

— *întreg* reprezintă un număr zecimal (cu valori între 1 și 99, valoarea implicită fiind 1) care indică de cîte ori rutina de trasare va semnala apelul modulelor și/sau segmentelor în cadrul proțiunii de program delimitată prin *nume1* și *nume2* ;

— opțiunile parametrului LN indică tipul apelurilor ce vor fi semnalate de rutina de trasare și anume : orice tip de apel (prin opțiunea ALL) ; orice tip de apel, cu excepția apelurilor modulelor din BSS (prin opțiunea SP) ; apelul numai a segmentelor (prin opțiunea SEG) ;

— opțiunea YES a parametrului CL indică imprimarea numelor punctelor de intrare.

**11.5.4. Vidajul memoriei.** O altă facilitate de punere la punct a programelor, oferită de monitor, este vidajul memoriei, care reprezintă conținutul zonei de memorie alocată unui program (conținut ce este înregistrat pe suportul fișierului de ieșire al sistemului). În prezent, majoritatea sistemelor de operare asigură această facilitate care, în funcție de modul în care este implementată, poate simplifica, prin informațiile furnizate, punerea la punct a programelor. În multe situații însă, volumul mare de informații furnizate de vidaj și modul în care acestea sînt expri-



Scanned with OKEN Scanner

mate (în general, în hexazecimal) fac ca intepretarea lor să fie dificilă (mai ales pentru programatorii în limbaje evoluate care, în general, nu au cunoştinţe temeinice despre structura şi funcţionarea sistemului de calcul).

În cazul sistemului de operare al calculatorului FELIX C-256 (sau IRIS-50), în funcţie de versiunea implementată (versiunea 14 sau 15), vidajul memoriei se poate prezenta sub forma :

$$\begin{array}{ccccccccccc} Y_1 & X_{11} & X_{12} & X_{13} & X_{14} & Y_2 & X_{21} & X_{22} & X_{23} & X_{24} \\ Y_3 & X_{31} & X_{32} & X_{33} & X_{34} & Y_4 & X_{41} & X_{42} & X_{43} & X_{44} \\ Y_{(n-1)} & X_{(n-1)1} & X_{(n-1)2} & X_{(n-1)3} & X_{(n-1)4} & Y_n & X_{n1} & X_{n2} & X_{n3} & X_{n4} \end{array}$$

sau sub forma

$$\begin{array}{l} Y_1\ X_{11}\ X_{12}\ X_{13}\ X_{14}\ X_{15}\ X_{16}\ X_{17}\ X_{18}\quad C_{11}\ C_{12}\ C_{13}\ C_{14}\ C_{15}\ C_{16}\ C_{17}\ C_{18} \\ Y_2\ X_{21}\ X_{22}\ X_{23}\ X_{24}\ X_{25}\ X_{26}\ X_{27}\ X_{28}\quad C_{21}\ C_{22}\ C_{23}\ C_{24}\ C_{25}\ C_{26}\ C_{27}\ C_{28} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ Y_n\ X_{n1}\ X_{n2}\ X_{n3}\ X_{n4}\ X_{n5}\ X_{n6}\ X_{n7}\ X_{n8}\quad C_{n1}\ C_{n2}\ C_{n3}\ C_{n4}\ C_{n5}\ C_{n6}\ C_{n7}\ C_{n8} \end{array}$$

unde $Y_i$ reprezintă adresa unei zone-memorie alcătuită dintr-o succesiune de patru, respectiv, opt cuvinte-memorie, în care se găsesc memorate informaţiile $X_{ij}$ ($j=1, 4$, respectiv, $j=1, 8$); $Y_j$ reprezintă succesiuni de şase cifre hexazecimale, iar $X_{ij}$ succesiuni de opt cifre hexazecimale; $C_{ij}$ ($j=1, \ldots, 8$) reprezintă succesiuni de patru caractere EBCDIC ce corespund informaţiilor $X_{ij}$.

Tipărirea informaţiilor $Y_i$, $X_{ij}$ şi $C_{ij}$ este precedată de tipărirea conţinutului registrului de stare a programului – ETAT PROGRAMME – şi al registrelor generale – REGISTRES (vezi programul 11.1).

Vidajul memoriei poate reprezenta conţinutul întregii zone de memorie alocată programului sau conţinutul unei părţi din această zonă şi poate fi obţinut implicit, în cazul detectării unei erori în faza de execuţie a programului, sau explicit, în cazul în care este solicitat de programator (în acest caz, vidajul poate fi obţinut la sfîrşitul normal sau anormal al fazei de execuţie).

Informaţiile conţinute de vidajul memoriei, furnizat de monitor, consemnează starea programului la sfîrşitul (normal sau anormal) execuţiei acestuia. Editorul de legături oferă însă posibilitatea obţinerii unui vidaj parţial al zonei de memorie alocată programului, în diferite momente ale execuţiei (momente ce corespund trecerii controlului execuţiei prin diferite puncte ale programului), prin inserarea în program a unei rutine speciale de inventariere. Inserarea acestei rutine poate fi solicitată de programator prin cartela XTRACT, al cărei format este :

$$\text{XTRACT } \textit{adr1}\ [,\ \text{DUMP} : (\textit{adr2},\ \textit{adr3},\ \left\{\begin{matrix}\text{X}\\ \text{C}\\ \text{A}\\ \text{K}\end{matrix}\right\})\ ]\ [,\ \text{TITL} : \text{'şir'}]\left[,\left\{\begin{matrix}\text{TERM}\\ \text{ABRT}\end{matrix}\right\}\right]$$

## Programul 11.1

```
*          JOB P111,AN:IS05,PN:FELIX
*          SEG FOR1
*          COMPILE FORTRAN,OBL,MAP
FORTRAN   STARTED
```

```
FORTRAN 00.00                                                    P111        20/05/77   22.22.38

      * SEGMENT
      * DEFINE FILE A2(DVT:MT,RCF:FB,RCS:8,BFS:80,SFD)=1
            COMMON/F8/F4(10)
            INTEGER F1,F3,F9
            DIMENSION F2(6),F10(10)
            F1=0
          1 READ(105,2,END=4)F2,F3,(F4(F9),F9=1,F3)
          2 FORMAT(5A4,A2,I2,10F3.1)
            CALL F6(F3,F5)
            WRITE(1)F3,F5
            DO 3 F9=1,F3
          3 F10(F9)=F4(F9)
            F11=F3
            CALL COB(F1,F2,F11,F10)
            F1=1
            GO TO 1
          4 F1=2
            CALL COB(F1)
            CALL A6
            END
```

```
0000  00000004  00000084  0000006C  00000000  00000069  00000000
0018  FFFFFFFF  00000000  00000001  00000028
0028  00080000  E2C1F206  12000000  23000100  00000000  00000000
0040  00000000  00000000  00000000  00000000  00000000  02020050
0058  00000000  00080007  00000000  00000000  00000006  00000000
0070  00000000  00000000  00000000  00000000  00000000

010C  4DF5C1F4  6BC1F26B  C9F26BF1  F0C6F348  F15D4040
0178  00000008  02000124  0400016C  00000000  00000010  02000120
0190  04000128  04000170  04000140  00000000  00000004  02000120
01A8  00000000  00000000

01B0  0 7 9 0 0A 0000          LD4,9       F8
01B4  0 0 1 0 2F 0000          BSG,1       I%INIT

LIGNE:  6
01B8  0 0 0 0 2A 0000          LD4I,0      0
01BC  0 6 0 0 5A 0120          ST4,0       F1

LIGNE:  7
01C0  0 6 1 0 2A 010C      1S  LD4I,1      2S
01C4  0 0 0 0 2A 0069          LD4I,0      10S
01C8  0 6 3 0 2A 0290          LD4I,3      4S
01CC  0 0 2 0 2A 0000          LD4I,2      0
01D0  0 0 1 0 2F 0000          BSG,1       I%RDF8R8
01D4  0 6 0 0 2A 0128          LD4I,0      F2
01D8  0 0 1 0 2A 0006          LD4I,1      6
01DC  0 0 1 0 2F 0000          BSG,1       I%USAR
01E0  0 6 0 0 2A 0124          LD4I,0      F3
01E4  0 0 1 0 2F 0000          BSG,1       [illegible]
01E8  0 0 0 0 2A 0001          LD4[illegible]
01EC  0 6 0 0 5A 0168              [illegible]  %
01F0  0 6 7 0 0A 0168
01F4  0 0 7 0 21 [illegible]       LD4I,2      4TEMP+52
01F8  0 0 [illegible]  0000        BSG,1       A6
01FC  [illegible] 0 0A 0000        LD4,14      %

LIGNE:  20
0280  0 0 0 0 2A 0000          LD4I,0      0
0284  0 0 0 0 2F 0000          BSG,0       I%STOP
```

Scanned with OKEN Scanner

| ETIQ | ADR. | ETIQ | ADR. | ETIQ | ADR. | ETIQ | ADR. | ETIQ | ADR. |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 01C0 | 2 | 010C | 3 | 0248 | 4 | 0290 | | |

| SYMBOLE | TYPE | CLAS | ADRESSE | LONG |
|---|---|---|---|---|
| A6 | R | SP | | |
| F10 | R | T | 0140 P | 28 |
| F3 | I | S | 0124 P | 4 |
| F6 | R | SP | | |
| I%INIT | | SP | | |
| I%ITOR | | SP | | |
| I%USAR | | SP | | |
| C0B | R | SP | | |
| F11 | R | S | 0170 P | 4 |
| F4 | R | T | 0000 C | 28 |
| F9 | I | S | 0168 P | 4 |
| I%IOI | | SP | | |
| I%RDFORO | | SP | | |
| I%WRBIN | | SP | | |
| F1 | I | S | 0120 P | 4 |
| F2 | R | T | 0128 P | 18 |
| F5 | R | S | 016C P | 4 |
| I%ENDIOL | | SP | | |
| I%IOR | | SP | | |
| I%STOP | | SP | | |

VARIABLES LOCALES ( 54 OCTETS )

0120 F1 0124 F3 0128 F2 0140 F10 0168 F9 016C F5
0170 F11

COMMON /F0 / ( 28 OCTETS

0000 F4

SOUS-PROGRAMMES DEMANDES :

C0B A6 I%INIT I%RDFORO I%USAR I%IOI I%IOR
I%ENDIOL I%WRBIN I%ITOR I%STOP F6

Scanned with OKEN Scanner

```
FORTRAN 00.00                                                    P111        20/05/77   22.22.38

LONGUEUR DU MODULE-OBJET PROGRAMME (F%MDATA ) :  2B8 OCTETS

                    F8              TYPE   C    LONGUEUR    0028 (00040)

          MODULE    F%MDATA         TYPE   P    LONGUEUR    02B8 (00696)

..... FIN DE COMPILATION (PLUS HAUT NIVEAU D'ERREUR RENCONTRE = 0)                22.22.44

.         SEG FOR2
.         COMPILE FORTRAN,OBL,MAP

FORTRAN 00.00                                                    P111        20/05/77   22.22.45

          * SEGMENT
                SUBROUTINE F6(F3,F5)
              . COMMON/F8/F4(10)
                INTEGER F3,F7
                F5=0.
                DO 4 F7=1,F3
              4 F5=F5+F4(F7)
                RETURN
                END

FORTRAN 00.00                                                    P111        20/05/77   22.22.45

          MODULE    F8              TYPE   C    LONGUEUR    0028 (00040)

          MODULE    F6              TYPE   P    LONGUEUR    0080 (00128)
```

```
*       COMPILE C0B0L,0BL,MAP
C0B0L   STARTED

0001            ID DIVISION.
0002            PROGRAM-ID. C08(
0003            ENVIRONMENT DIVISION.
0004            CONFIGURATION SECTION.
0005            INPUT-OUTPUT SECTION.
0006            FILE-CONTROL.
0007                SELECT C1 ASSIGN TO BRD.
0008                SELECT C2 ASSIGN TO SYSOUT.
0009            DATA DIVISION.
0010            FILE SECTION.
0011            FD  C1 RECORDING V
0012                LABEL RECORD STANDARD
0013                BLOCK 3 TO 7 RECORDS.
0014            01  C3.
0015              02 C4        PIC X(22).
0016              02 C5        PIC 99.
0017              02 C6        PIC 9999V9
0018                           OCCURS 2 TO 10 DEPENDING C5.
0019            FD  C2 RECORDING F
0020                LABEL RECORD OMITTED.
0021            01  C7.
0022              02 C8        PIC X.
0023              02 C9        PIC Z9.
0024            LINKAGE SECTION.
0025            77  C11        PIC S9(5) COMP.
0026            77  C12        PIC S9(5) COMP.
0027            01  C13.
0028              02 C14       COMP-1.OCCURS 10.;
0029            01  C15        PIC X(22).
0030            01  C16        PIC S9(5). COMP.
0031            PROCEDURE DIVISION USING C11, C15, C12, C13.
0032            1.  IF C11 = 0 OPEN OUTPUT C1 C2 GO TO 2.
0033                  IF C11 = 2 CLOSE C1 C2 GO TO 4.
0034            2.      COMPUTE C12 = - C12
0035                    PERFORM 5 VARYING C12 FROM C12 BY 1 UNTIL C12 = 0
0036                    MOVE  C15 TO C4
0037                    MOVE  C12 TO C5 C9
0038                    WRITE C3
0039                    WRITE C7 AFTER 1.
0040            4.  EXIT PROGRAM.
0041            5.  MOVE C14 (C12) TO C6 (C12).
```



Scanned with OKEN Scanner

## GEOGRAPHIE DES DONNEES

| | NLSR | CAT | NIV | ADRESSE | LONGUEUR | TYPE | IDENTIFICATEUR |
|---|---|---|---|---|---|---|---|
| FILE SECTION | | | | | | | |
| | 11 | FICH | | | | | C1 |
| | 14 | ART | 01 | 0000 | 74 | GR | C3 |
| | 15 | | 02 | 0000 | 22 | AN | C4 |
| | 16 | | 02 | 0016 | 2 | DE | C5 |
| | 17 | | 02 | 0018 | 5 | DE | C6 |
| | 19 | FICH | | | | | C2 |
| | 21 | ART | 01 | 0000 | 3 | GR | C7 |
| | 22 | | 02 | 0000 | 1 | AN | C8 |
| | 23 | | 02 | 0001 | 2 | NE | C9 |
| WORK SECTION | | | | | | | |
| | 24 | | 77 | 0032 | 2 | BHW | TALLY |
| | 24 | | 77 | 0034 | 2 | BHW | RETURN-CODE |
| | 24 | | 77 | 0038 | 4 | BW | SORT-FILE-SIZE |
| | 24 | | 77 | 003C | 4 | BW | SORT-CORE-SIZE |
| | 24 | | 77 | 0040 | 4 | BW | SORT-MODE-SIZE |
| | 24 | | 77 | 0044 | 2 | BHW | SORT-RETURN |
| LINK SECTION | | | | | | | |
| | 25 | | 77 | 0060 | 4 | BW | C11 |
| | 26 | | 77 | 0068 | 4 | BW | C12 |
| | 27 | | 01 | 0000 | 40 | GR | C13 |
| | 28 | | 02 | 0000 | 4 | FS | C14 |
| | 29 | | 01 | 0000 | 22 | AN | C15 |
| | 30 | | 01 | 0000 | 4 | BW | C16 |

## GEOGRAPHIE DES FICHIERS

| NOM DU FICHIER | IDEX | ORG | PRM | ACM | RAM | RCF | BFN | BFS | RCS | RCL | ATOF | ARCA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1 | B | SEQ | OUTP | SEQ | MOV | VAR | 2 | 561 | 74 | | 0168 | 0078 |

Scanned with OKEN Scanner

| | ADRESSE | LONGUEUR | VALEUR |
|---|---|---|---|
| CONSTANTES | 0000 | 0032 | F00FFF5000010000000000000000000C3D9C4C24B0000 0000000 00000000000000000000 |
| VALEUR INITIALE | 0034 | 0002 | 0000 |
| VALEUR INITIALE | 0038 | 0004 | 00000000 |
| VALEUR INITIALE | 003C | 0004 | 00000000 |
| VALEUR INITIALE | 0040 | 0004 | 00000000 |
| VALEUR INITIALE | 0044 | 0002 | 0000 |
| T.D.F. | 0168 | 0058 | E2C2F2071200000000000000010000000000000000000000000 000 0202023100000000004A01000000000000000000000000000000 000 0000000000000000 |
| MASQUES D'EDITION | 01C0 | 0002 | 2120 |
| TABLE DES SORTIES DE PERFORM | 01D0 | 0004 | TOUT A S'FF' |

| *LIG.SOURC | *CODE VERBE | *CPT.ORDI | *INSTRUCTION | | *INSTRUCTION DECODEE | | *SYMBOLIQUE |
|---|---|---|---|---|---|---|---|
| 0032 | | 0000 | 7E0A0000 | X | LD4 ,14 | 15.0000 | |
| | | 0004 | 50090032 | | LDH2, 0 | 13.0032 | |
| | | 0008 | 60590032 | | STH2, 0 | 14.0032 | |
| | | 000C | 6A5B0048 | | ST8 ,10 | 14.0048 | C%ZM000 |
| | | 0010 | 6C5B0050 | | ST8 ,12 | 14.0050 | C%ZM000 |
| | | 0014 | 685B0058 | | ST8 , 8 | 14.0058 | C%ZM000 |
| | | 0018 | 032A0014 | | LD4I, 3 | 0014 | |
| | | 001C | 62490060 | | MVSR, 2 | 14.0060 | C%ZM000 |
| | | 0020 | 7E0A0000 | X | LD4 ,14 | 15.0000 | |
| | | 0024 | 7D0A0000 | X | LD4 ,13 | 15.0000 | |
| | | 0028 | 5F410000 | | LDTM,15 | 13.0000 | |
| | | | | | | C8B%0201 | |
| | IF | 002C | D00A0060 | | LD4 , 0 | *13.0060 | C11 |
| | | 0030 | 6834006C | | BCF , 8 | 14.006C | |
| | OPEN | 0034 | 522A0168 | | LD4I, 2 | 13.0168 | C1 |
| | | 0038 | 525A0158 | | ST4 , 2 | 13.0158 | C1 |
| | | 003C | 552A0188 | | LD4I, 5 | 13.0188 | C1 |
| | | 0040 | 555A015C | | ST4 , 5 | 13.015C | |
| | | 0044 | 550C015C | | LAS , 5 | 13.015C | |
| | | 0048 | 032A0000 | | LD4I, 3 | 0000 | |
| | | 004C | 012F0000 | X | BSG , 1 | 0000 | C%0070 |
| | | 0050 | 7E0A0000 | X | LD4 ,14 | 15.0000 | |
| | | 0054 | 012F0000 | X | BSG , 1 | [illegible] | |
| | | 0058 | 7E0A0000 | X | LD4 ,14 | 0005 | |
| | | 005C | 032A0000 | | [illegible] 3 | 13.01C8 | C%ZM001 |
| | | 0060 | 012F0000 | | SHD , 0 | 0003 | |
| | | 0064 | 7F[illegible]0 | | LD1I, 0 | 0040 | |
| | | 0068 | 012A0002 | | LD4I, 1 | 0002 | |
| 0033 | IF | [illegible] | 012F0000 | X | BSG , 1 | 0000 | C%0121 |
| | | 0130 | 500200D1 | | DATA,4,4 | | C9 |
| | | 0134 | 7E0A0000 | X | LD4 ,14 | 15.0000 | |
| 0038 | WRITE | 0138 | 012F0000 | X | BSG , 1 | 0000 | C%0091 |
| | | 013C | 5002008E | | DATA,4,4 | | C3 |
| | | 0140 | 7E0A0000 | X | LD4 ,14 | 15.0000 | |
| | | 0144 | 07290005 | | LD2I, 7 | 0005 | C3 |
| | | 0148 | 07310018 | | MPU2, 7 | 0018 | |
| | | 014C | 040A001C | | LD4 , 4 | 001C | |

| *LIG.SOURC | *CODE VERBE | *CPT.ORDI | *INSTRUCTION. | | *INSTRUCTION DECODEE | | * | *SYMBOLIQUE |
|---|---|---|---|---|---|---|---|---|
| | | 0150 | 04270018 | | AD4I, 4 | 0018 | C3 | |
| | | 0154 | 520A0158 | | LD4 , 2 | 13.0158 | C3 | |
| | | 0158 | 550A015C | | LD4 , 5 | 13.015C | | |
| | | 015C | 532A0078 | | LD4I, 3 | 13.0078 | C3 | |
| | | 0160 | 012F0000 | X | BSG , 1 | 0000 | C%0060 | |
| | | 0164 | 7E0A0000 | X | LD4 ,14 | 15.0000 | | |
| 0039 | WRITE | 0168 | 042A0004 | | LD4I, 4 | 0004 | C7 | |
| | | 016C | 532A00CF | | LD4I, 3 | 13.00CF | C7 | |
| | | 0170 | 012F0000 | X | BSG , 1 | 0000 | C%0068 | |
| | | 0174 | 7E0A0000 | X | LD4 ,14 | 15.0000 | | |
| | | 0178 | 012F0000 | X | BSG , 1 | 0000 | C%0051 | |
| | | 017C | 7E0A0000 | X | LD4 ,14 | 15.0000 | | |
| 4 | | | | | | C0B%0207 | | |
| 0040 | EXIT | 0180 | 5A0B0048 | | LD8 ,10 | 13.0048 | C%ZM001 | |
| | | 0184 | 580B0058 | | LD8 , 8 | 13.0058 | [illegible] | |
| | | 0188 | 50090032 | | LDH2, 0 | [illegible] | C12 | |
| | | 018C | 5C0B0050 | | LD8 | 13.0068 | | |
| | | 0190 | 50390032 | | NPU2, 7 | 9.0002 | C12 | |
| | | 0194 | 8038[illegible] | X | BSG , 1 | 0000 | C%0100 | |
| 5 | | | [illegible]0088 | | DATA,4,4 | | C6 | |
| 0041 | MOVE | 0198 | 00FF0000 | | DATA,4,4 | | | |
| | | [illegible] | 7E0A0000 | X | LD4 ,14 | 15.0000 | | |
| | | 01CC | 57AA008F | | LD4I, 0 | 13.008F, 7 | C6 | |
| | | 01D0 | 8F400000 | | STTM,15 | * .0000 | | |
| | | 01D4 | 590C01D0 | | LAS , 9 | 13.01D0 | | |
| | | 01D8 | 14340004 | | BCF , 4 | 9.0004 | | |
| C%ETFNPR | | | | | | C0B%0209 | | |

ETAT RECAPITULATIF DU PROGRAMME: C0B

SEGMENT C0B%%%00 NUMERO 01

MODULE D C0B%%%00 LONGUEUR 01D4

MODULE C C0B%%%B2 LONGUEUR 0462

MODULE P C0B%%%01 LONGUEUR 01DC

FIN NORMALE DE COMPILATION

```
.          ENTSG A6
.          COMPILE ASSIRIS,MAP
ASSIRIS  STARTED

ASSIRIS 15.A     20/05/77  22.23

                                  1    A1        CSECT
                                  2              DEF      A6
0000    E2C3F107   A              3    A3        SFD      C1,DVT:RD,RCF:VAR,RCS:74,BFS:561,MCD:(GT),EOF:A8,
0004    01000000   A              4                       PRM:INP
0008    00000000   A
000C    00000000
0010    00000000   A
0014    00000000   A
0018    00000000   A
001C    00000000   A
0020    00000000   A
0050    00000000   A
0054    00000000   A
0058    E25CF302   A              5    A2        SFD      *3,DVT:CP,RCS:24,BFS:256,MCD:(PT),PRM:OUT
005C    02000000   A
0060    00000000   A
0064    00000000   A
0068    00000000   A
006C    00000000   A
0070    00000000   A
        00000000   A
00A8    00000000   A
00AC    00000000   A
00B0                              6    A4        RES      79
00FF                              7    A5        RES      24
0118    F10A0000   A              8    A6        OPEN     A2,A3
011C    622A012C
0120    002A0000   A
0124    682A0138
0128    01880010   A
012C    00000002   A
0130    00000058
0134    00000000
```

```
0138    622A0000                 9    A7        GET       A3,RCA:A4
013C    632A00B0
0140    022800A1   A
0144    E8390020
0148    602A00B0                10              LD4I,0    A4
014C    600A0190                11              LD4,0     L(24)
0150    604900FF                12              MVSR,0    A5
0154    622A0058                13              PUT       A2,RCA:A5
0158    632A00FF
015C    022800A2   A
0160    E8390078
0164    60380138                14              BRU       A7
0168    F10A0000   A            15    A8        CLOSE     A2,A3
016C    622A017C
0170    002A0001   A
0174    682A0188
0178    01B80010   A
017C    00000002   A
018C    00000058
0184    00000000
0188    042E0000   A            16              TERM
                                17              END
0190    00000018   A
0194    00000000   A
```

ASSIRIS 15.A  20/05/77  22.23

***DENSITE DE COMMENTAIRES  0

***NOMBRE DE CONSTANTES D'ADRESSE DE TYPE RA :  5

MODULE  A1  TYPE  P  LONGUEUR  0198

ASSIRIS 15.A  20/05/77  22.23

| ***SYMBOLE | SECT | VALEUR | TYPE |
|---|---|---|---|
| A1 | P 1 | 00000000 | D |
| A2 | P 1 | 00000058 | |
| A3 | P 1 | 00000000 | |
| A4 | P 1 | 000000B0 | |
| A5 | P 1 | 000000FF | |
| A6 | P 1 | 00000118 | D |
| A7 | P 1 | 00000138 | |
| A8 | P 1 | 00000168 | |

```
          TREE F0R1+F8(F0R2(C0B%%%00))
          LINK
LINK      STARTED


LINK    . 15.06.03   20/05/77  22H24M44S


SEGMENT   F0R1        N0   1     IMPLANTATION     0
          M0DULE  F%MDATA         IMPLANTATION    70
          M0DULE  TDFSYSUT        IMPLANTATION   328
          M0DULE  TDFSYSIN        IMPLANTATION   398
          M0DULE  I%INIT          IMPLANTATION   410
          M0DULE  I%READ          IMPLANTATION   8C0
          M0DULE  I%USAI          IMPLANTATION   AE0
          M0DULE  I%I0I           IMPLANTATION   C80
          M0DULE  I%END10L        IMPLANTATION   D50
          M0DULE  I%WRBIN         IMPLANTATION   D78
          M0DULE  I%IT0R          IMPLANTATION  1018
          M0DULE  I%ST0P          IMPLANTATION  1048
          M0DULE  F%ERR1          IMPLANTATION  1160
          M0DULE  ST0PRUN         IMPLANTATION  1288
          M0DULE  I%LECARI        IMPLANTATION  12F8
          M0DULE  I%F0RMA         IMPLANTATION  15E8
          M0DULE  I%D0NNEE        IMPLANTATION  2C38
          M0DULE  I%ECRARI        IMPLANTATION  2CB0
                                  LONGUEUR DU SEGMENT 3018

SEGMENT   F8          N0   2     IMPLANTATION  3018
          M0DULE  F8              IMPLANTATION     0
                                  LONGUEUR DU SEGMENT   28

SEGMENT   C0B%%%B2    N0   3     IMPLANTATION  3040
          M0DULE  C0B%%%B2        IMPLANTATION     0
                                  LONGUEUR DU SEGMENT  468

SEGMENT   F0R2        N0   4     IMPLANTATION  34A8
          M0DULE  Fe              IMPLANTATION     8
                                  LONGUEUR DU SEGMENT   88

SEGMENT   C0B%%%00    N0   5     IMPLANTATION  353C
          M0DULE  C0B%%%01        IMPLANTATION    40
          M0DULE  A1              IMPLANTATION   220
          M0DULE  8PCLTYP         IMPLANTATION   3B8
          M0DULE  C%BBD4          IMPLANTATION   4C8
          M0DULE  C0121           IMPLANTATION   588
          M0DULE  C%DDB2          IMPLANTATION   660
          M0DULE  UTWRTPCH        IMPLANTATION   758
          M0DULE  C%WRITE2        IMPLANTATION   7E0
          M0DULE  C%FFIDE         IMPLANTATION   838
          M0DULE  C%WRITEA        IMPLANTATION   8D0
          M0DULE  C%WRITEB        IMPLANTATION   908
          M0DULE  C%BBD8          IMPLANTATION   960
          M0DULE  C%FIBI          IMPLANTATION   A58
                                  LONGUEUR DU SEGMENT  B28

SEGMENT   C0B%%%00    N0   6     IMPLANTATION  4058
          M0DULE  C0B%%%00        IMPLANTATION     0
                                  LONGUEUR DU SEGMENT  108
```

Scanned with OKEN Scanner

```
LINK      15.06.03    20/05/77   22H24M45S

                    IMPLANT. APRES TRAITEMENT OPTION FMS

  SEGMENT   FOR1      NO   1    IMPLANTATION      0
                                     LONGUEUR DU SEGMENT 4E88

  SEGMENT   F8        NO   2    IMPLANTATION  4E88
                                     LONGUEUR DU SEGMENT   28

  SEGMENT   COB%%%B2  NO   3    IMPLANTATION  4EB0
                                     LONGUEUR DU SEGMENT  468

  SEGMENT   FOR2      NO   4    IMPLANTATION  5318
                                     LONGUEUR DU SEGMENT   88

  SEGMENT   COB%%%00  NO   5    IMPLANTATION  53A0
                                     LONGUEUR DU SEGMENT 21D0

  SEGMENT   COB%%%00  NO   6    IMPLANTATION  757C
                                     LONGUEUR DU SEGMENT  228


LINK      15.06.03    20/05/77   22H24M45S

          0     ERREUR  EN EDITION DE LIENS
        ADRESSE DE LANCEMENT                    220
        LONGUEUR PLUS GRANDE BRANCHE           7798
        LONGUEUR DU PROGRAMME EDITE            7798

              PLUS HAUT NIVEAU D'ERREUR RENCONTRE  N=0   (PAS D'ERREUR)
```

```
INIT DV:MT1,VB1123456
ASSIGN A,DV:MT1
LABEL A,FN:'FBAND'
INIT DV:RD1,VB1123456
ASSIGN B,DV:RD1
LABEL B,FN:'FDISC',AM:ANY,SZ:10
ASSIGN C,DV:RD1
LABEL C,FN:'FDISC'
RUN UD:CP
STARTED
```

VIDAGE PARTITION 02

B2EGF0432R4>RCS TDF.

ETAT PROGRAMME 00024ADC F0140000

| REGISTRES | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
|---|---|---|---|---|---|---|---|---|
| | 00024AFF | 00024ADC | FF027800 | 00027710 | 0000016C | 80027820 | 00040004 | 00000154 |
| | R8 | R9 | R10 | R11 | R12 | R13 | R14 | R15 |
| | 00025C54 | 04020200 | 00025FEC | FF0213CE | 00000788 | 00027698 | 000254C8 | 0002010C |

VIDAGE MEMOIRE

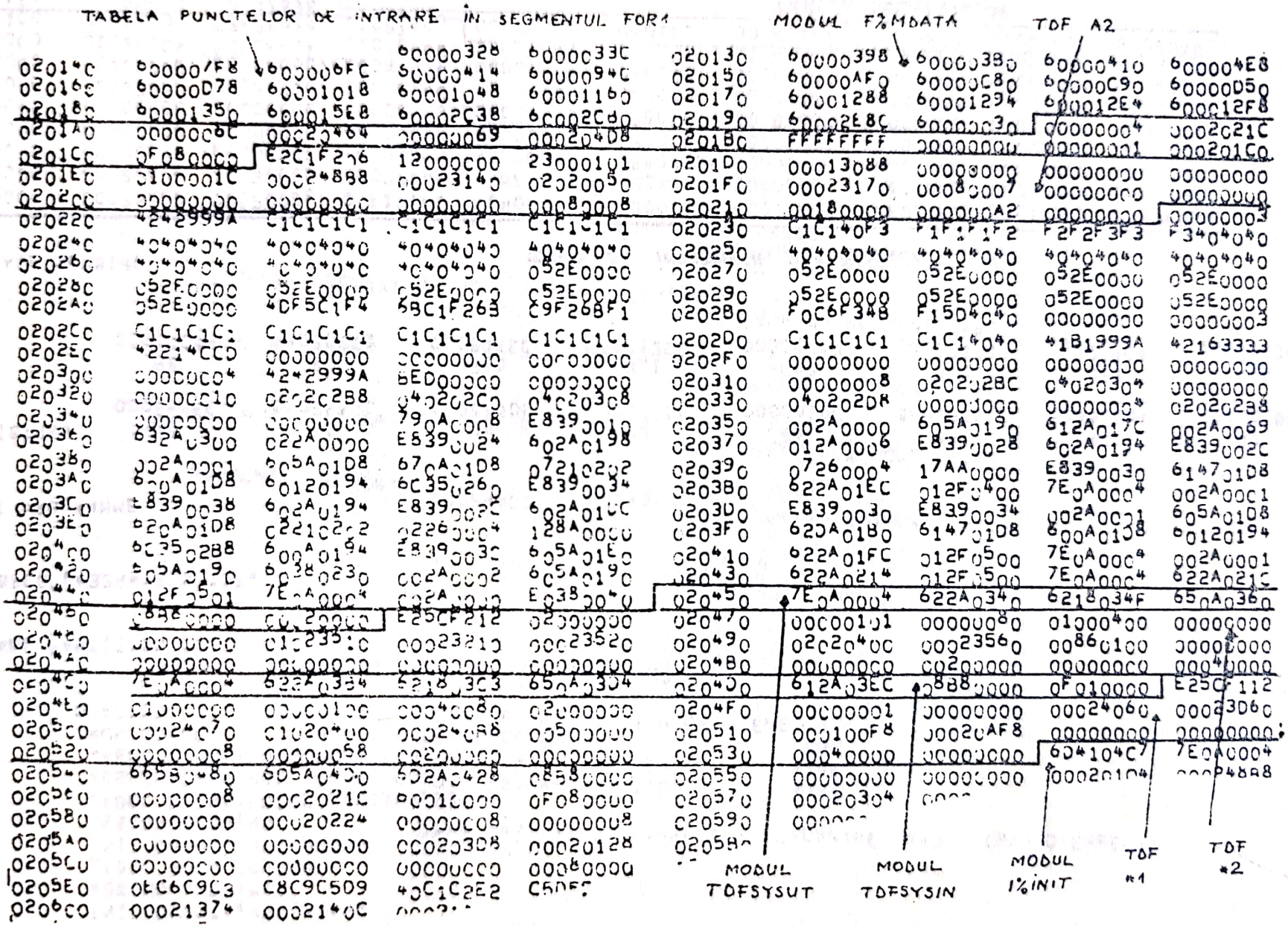
ZONA DE INFORMAȚII GENERALE

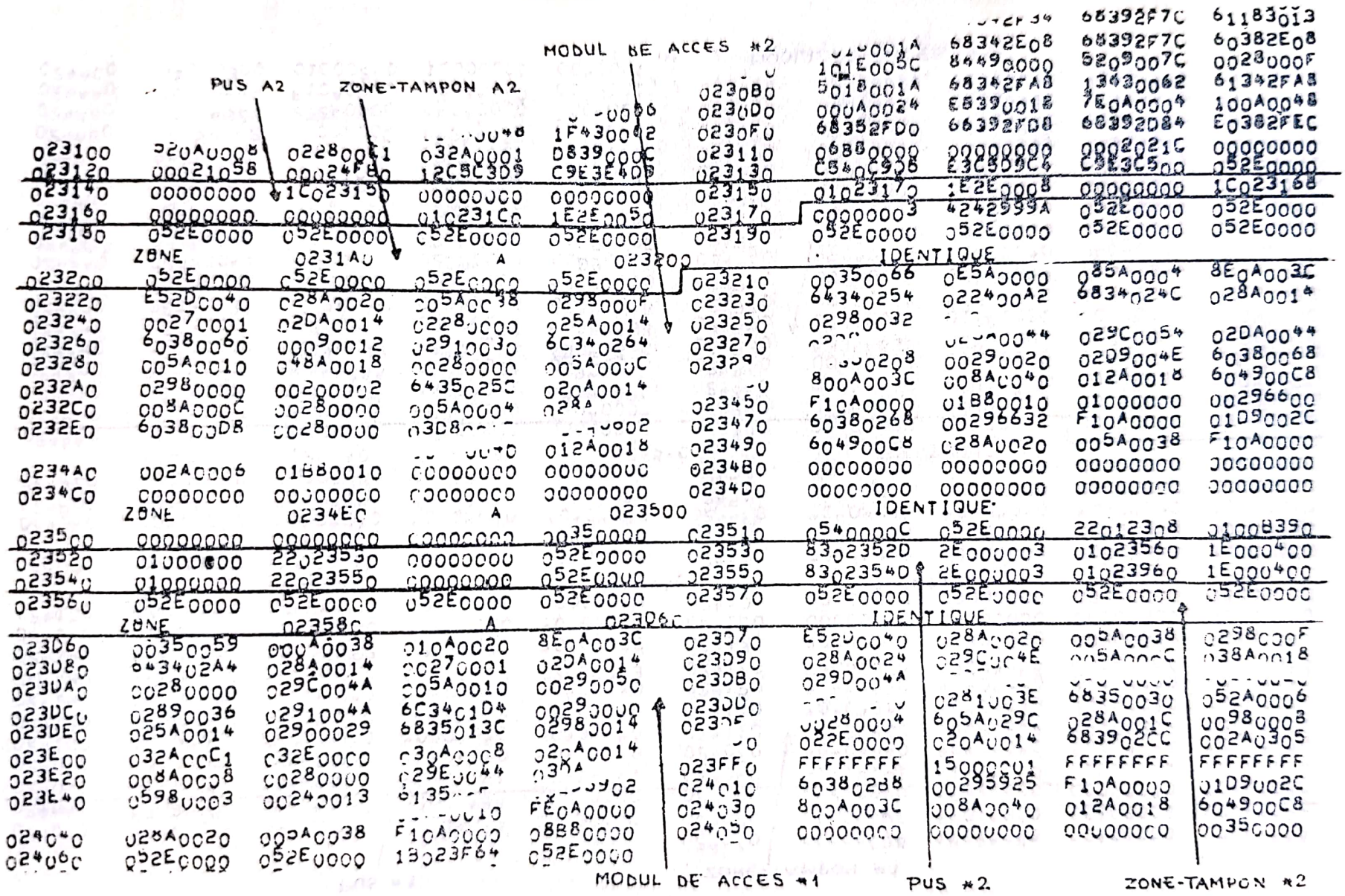
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 020000 | 00024AD0 | 000200D8 | D7F1F1F1 | 40404040 | 020010 | 0000FF00 | F2F0F1F0 | F5F1F7F7 | F1F4F000 |
| 020020 | 0002FFFF | 00030000 | 000278BF | 400278BF | 020030 | 00000B0A | 00000000 | 00020048 | 0403040F |
| 020040 | 00020098 | 00000000 | 00000000 | 00000000 | 020050 | 00000000 | 00000000 | 00000000 | 00000000 |
| 020060 | 00000000 | 00000000 | 00000000 | 00000000 | 020070 | 00000000 | 00000000 | 00000000 | 00000000 |
| 020080 | 00000000 | 00000000 | 00000000 | 00000000 | 020090 | 00000000 | 00000000 | 000254C8 | 00025C54 |
| 0200A0 | A2027800 | 00027710 | 0000016C | 80027820 | 0200B0 | 00024A38 | 00024839 | 00024D5C | 0000000A |
| 0200C0 | 405CF368 | 00000000 | 00024C18 | 00002738 | 0200D0 | 00020098 | 00024858 | 04001A40 | C1F20024 |
| 0200E0 | 010201C4 | 5CF24024 | 01020468 | 5CF18024 | 0200F0 | 01020400 | C3F1C024 | 00000000 | 5CF30028 |
| 020100 | 00000000 | C2F24028 | 01027800 | 00020000 | 020110 | 00020128 | 00024FB0 | 00024FD8 | 00025440 |
| 020120 | 00025400 | 00027698 | | | | | | | |

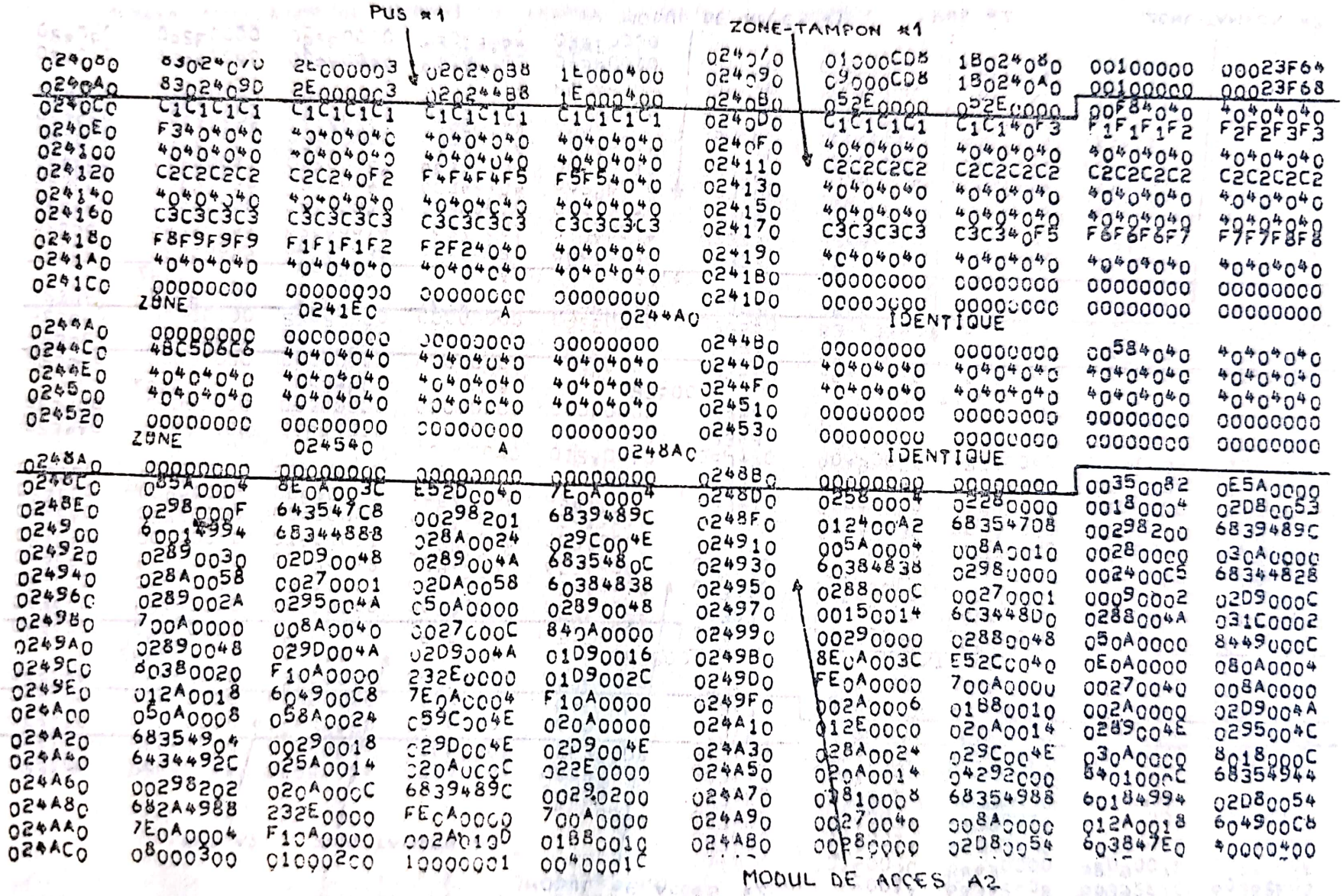
TABELA SEGMENTELOR

Scanned with OKEN Scanner

TABELA PUNCTELOR DE INTRARE IN SEGMENTUL FOR4

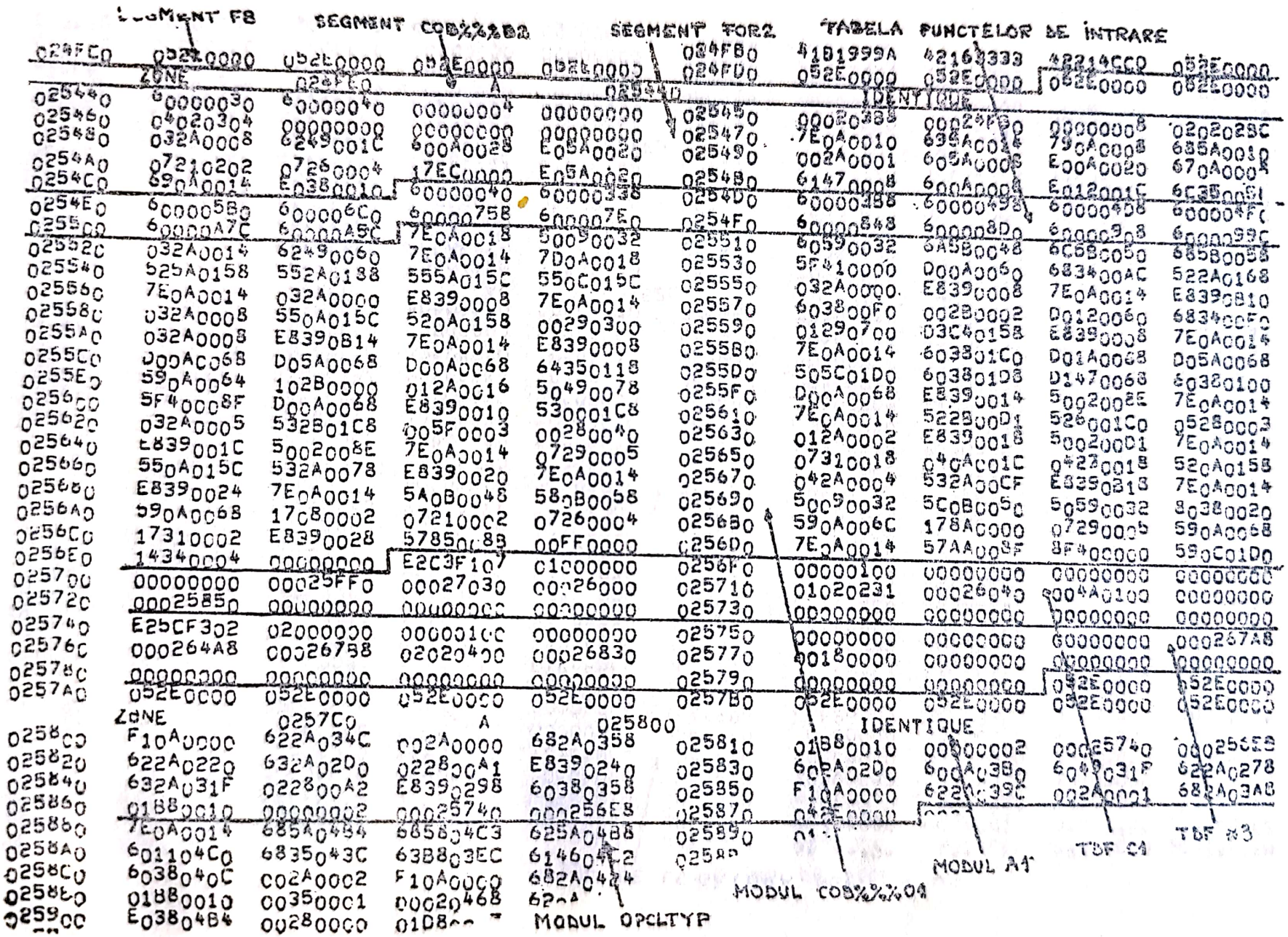


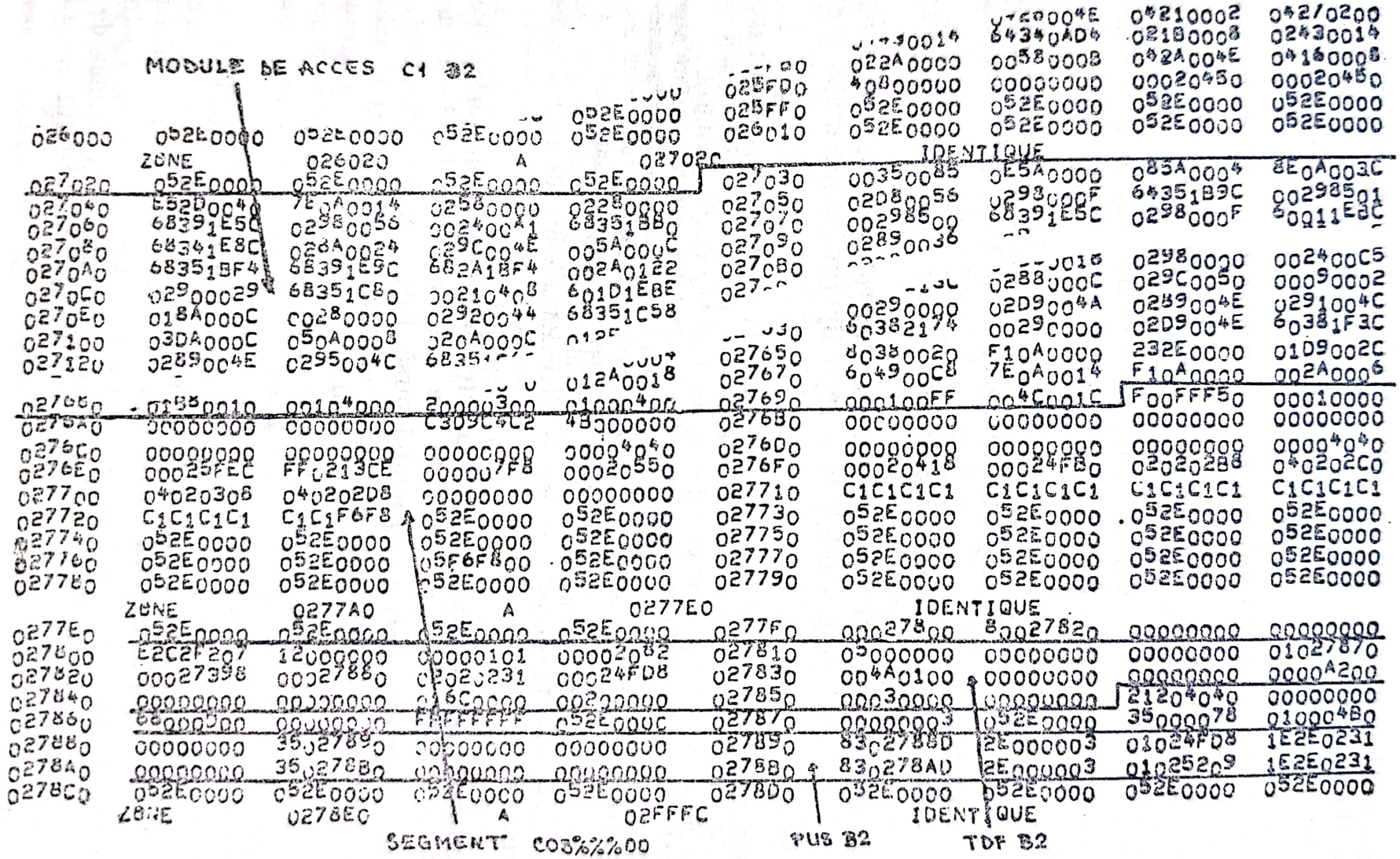

MODUL DE ACCES #2
PUS A2
ZONE-TAMPON A2
ZONE
A
IDENTIQUE
MODUL DE ACCES #1
PUS #2
ZONE-TAMPON #2


PUS #1
ZONE-TAMPON #1
ZONE
0241E0
A
0244A0
IDENTIQUE
ZONE
024540
A
0248AC
IDENTIQUE
MODUL DE ACCES A2

MODUL DE LEGĂTURĂ SGF

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | 024AD0 | 00024AD4 | 01AA002F | 092E0000 | 002A0548 |
| 024AE0 | 0B2E0000 | 800A0004 | 80380000 | 000258EC | 024AF0 | 002E0000 | 00024B13 | 04024AFF | 843200C2 |
| 024B00 | F2C5C7C6 | F8F4F3F2 | D9F46ED9 | C3E240E3 | 024B10 | C4C64B15 | C6C4C9E2 | C3404040 | 40404040 |
| 024B20 | 40404040 | 40404040 | 40404040 | 40404040 | 024B30 | 40404040 | 40404040 | 40404040 | 40404040 |
| 024B40 | 40404040 | 40404040 | 40404040 | 40404040 | 024B50 | 40404040 | 40404040 | 40404040 | 404040FF |
| 024B60 | F0F0F0F1 | F0F0F0F0 | F0F0F0F0 | 00000003 | 024B70 | F0F00000 | 0000000A | C1000001 | 01010006 |
| 024B80 | 00024B80 | 00024AD0 | 000258E4 | 00000000 | 024B90 | 00020223 | 00023210 | 000254C8 | 00025C54 |
| 024BA0 | A2027800 | 00027710 | 0000016C | 80027820 | 024BB0 | 00164040 | 40404040 | 000E4040 | 47F1D7F1 |
| 024BC0 | F1F14000 | 0A034040 | 404040F0 | F1F0F0F0 | 024BD0 | F1F0F0F7 | F7F1F4F0 | F7F7F1F4 | F0401800 |
| 024BE0 | 03008120 | 00030100 | 00780005 | 27000000 | 024BF0 | 00000000 | 40404040 | 40404040 | 40404040 |
| 024C00 | 40404040 | 40404040 | 40404040 | 40404040 | 024C10 | C8C4D9F2 | C30231E5 | 004A0000 | 00404040 |
| 024C20 | 08404040 | 40404040 | 40404040 | 40404040 | 024C30 | C5D6C4E8 | 40404040 | 40404040 | 40404040 |
| 024C40 | 40404040 | 40404040 | 40404040 | 40404040 | 024C50 | 40404040 | 40404040 | 40404040 | 40403800 |
| 024C60 | 03404000 | 00054040 | 40404040 | 40404040 | 024C70 | 40404040 | 40404040 | 40404040 | 40404040 |
| 024C80 | 40404040 | 40404040 | 40404040 | 40404040 | 024C90 | FFFFFFF8 | 00027404 | 00027800 | 00027710 |
| 024CA0 | 0000016C | 80027820 | 843200C2 | FF027800 | 024CB0 | 40404040 | 40404040 | C2F24040 | 070135DA |
| 024CC0 | 002C0143 | 0BE14040 | 40404040 | 40404040 | 024CD0 | 40404040 | 40404040 | 40404040 | 40404040 |
| 024CE0 | 40404040 | 40404040 | 40404040 | 40404040 | 024CF0 | C6C4C9E2 | C3404040 | 40404040 | 40404040 |
| 024D00 | 40404040 | 40404040 | 40404040 | 40404040 | 024D10 | 40404040 | 40404040 | 40404040 | 40404040 |
| 024D20 | 40404040 | 40404040 | 40404040 | 40404040 | 024D30 | 40404040 | 40404040 | 404040F0 | F0F0F0F0 |
| 024D40 | F0F0F0F0 | F0F0F0F0 | F0F0F0F0 | F0F00000 | 024D50 | 0000000A | C1000000 | FF00FF00 | FF00FF00 |
| 024D60 | FF00FF00 | D9C4F140 | 40404040 | 40404040 | 024D70 | 40404040 | 40404040 | 40404040 | 40404040 |
| 024D80 | 40404040 | 40404040 | 40404040 | 40404040 | 024D90 | 40000000 | 00000000 | 00000000 | 00000000 |
| 024DA0 | 00000000 | 00000000 | 00000000 | 00000000 | 024DB0 | 40404040 | 40404040 | 40404040 | 00004040 |
| 024DC0 | 40404040 | 40404040 | 40404040 | 40404040 | 024DD0 | 40404040 | 40404040 | 40404040 | 40404040 |
| | ZONE | 024DE0 | A | 024E20 | | IDENTIQUE | | | |
| 024E20 | 40404040 | 40404040 | 40404040 | 40404040 | 024E30 | 40404040 | 40404040 | 404040F0 | F0F0F0F0 |
| 024E40 | F0F0F0F0 | F0F0F0F0 | F0F0F0F0 | F0F00000 | 024E50 | 00000000 | 00000000 | FF00FF00 | FF00FF00 |
| 024E60 | FF00FF00 | 00000000 | 00000000 | 00000000 | 024E70 | 00000000 | 00000000 | 00000000 | 00000000 |
| 024E80 | 00000000 | 00000000 | 00000000 | 00000000 | 024E90 | 00000000 | 00000000 | 00000000 | 00000000 |
| 024EA0 | 00000000 | 00000000 | 00000000 | 00000000 | 024EB0 | 40404040 | 40404040 | 40404040 | 00004040 |
| 024EC0 | 40404040 | 40404040 | 40404040 | 40404040 | 024ED0 | 40404040 | 40404040 | 40404040 | 40404040 |
| | ZONE | 024EE0 | A | 024F20 | | IDENTIQUE | | | |
| 024F20 | 40404040 | 40404040 | 40404040 | 40404040 | 024F30 | 40404040 | 40404040 | 404040F0 | F0F0F0F0 |
| 024F40 | F0F0F0F0 | F0F0F0F0 | F0F0F0F0 | F0F00000 | 024F50 | 00000000 | 00000000 | FF00FF00 | FF00FF00 |
| 024F60 | FF00FF00 | 00000000 | 00000000 | 00000000 | 024F70 | 00000000 | 00000000 | 00000000 | 00000000 |
| 024F80 | 00000000 | 00000000 | 00000000 | 00000000 | 024F90 | 00000000 | 00000000 | 00000000 | 00000000 |
| 024FA0 | 00000000 | 00000000 | 00000000 | 00000000 | | | | | |


SEGMENT FB
SEGMENT COB%%%B2
SEGMENT FOR2
TABELA FUNCTIILOR DE INTRARE
ZONE
A
IDENTIQUE
MODUL COB%%%04
MODUL OPCLTYP
MODUL A1
TBF #1
TBF #3


MODULE DE ACCES C1 B2
ZONE
A
IDENTIQUE
SEGMENT C03%%%00
PUS B2
TDF B2

unde : *adr1* reprezintă adresa instrucțiunii înaintea căreia va fi inserăt secvența de apel al rutinei inventariere (această adresă trebuie indicată sub forma $S \pm D$, unde $S$ reprezintă o definire externă a segmentului ce conține instrucțiunea de adresă *adr1*, iar $D$ o adresă relativă exprimată printr-un număr întreg sau o constantă hexazecimală formată din maximum 15 cifre binare) ;

— *adr2* și *adr3* reprezintă adresele de început și de sfîrșit ale zonei-memorie al cărei conținut este imprimat (aceste adrese, exprimate tot sub forma $S+D$, trebuie să aparțină aceluiași segment, care poate fi segmentul cu secvența de apel al rutinei de inventariere sau un segment de nivel superior acestuia) ;

— opțiunile X, C, A și K indică forma de imprimare : hexazecimală, în caractere, mixtă și, respectiv, hexazecimală, cu imprimarea simbolurilor recunoscute ;

— argumentul TITL indică, printr-un șir de maximum opt caractere, titlul ce va fi dat inventarului ;

— opțiunile TERM și ABRT indică sfîrșitul normal, respectiv anormal, al execuției programului, însă după tipărirea conținutului zonei de memorie.

**Observații.** Rutina de inventariere este inserată la sfîrșitul segmentului rădăcină.

Vidajul memoriei este imprimat ori de cîte ori, la execuție, este întîlnită instrucțiunea de adresă *adr1* (a cărei execuție are loc după imprimarea acestuia).

Informațiile furnizate de vidaj reprezintă conținutul registrului de stare a programului și al registrelor generale, adresele în cadrul programului și în cadrul segmentului, ale grupelor de cuvinte-memorie ce compun zona inventariată și conținutul acestora.

În cartela XTRACT pot figura maximum șase parametri DUMP, iar în absența acestora, este tipărit conținutul zonei-memorie alocate programului, începînd cu primul segment pînă la segmentul în care este inserată secvența de apel a rutinei de inventariere.

**Identificarea informațiilor furnizate de vidajul memoriei.** Pentru a identifica diferitele componente ale unui program în vidajul memoriei sînt necesare : lista programului sursă, lista programului obiect, geografia datelor, situațiile tipărite de programul traducător și de editorul de legături, precum și cunoașterea modului de implantare a programului în memorie (vezi 11.3). O sintetizare a modului de identificare a acestor componente este prezentată în cele ce urmează ; referirile concrete vizează componentele programului 11.1.

Primul cuvînt al registrului de stare a programului (ETAT PROGRAMME) reprezintă adresa instrucțiunii în limbaj-mașină care urma să fie executată sau la execuția căreia s-a semnalat eroarea. Primul cuvînt al tabelei segmentelor reprezintă adresa ZIG, iar următoarele, adresele segmentelor programului, conform ordinii atribuite segmentelor de editorul de legături (de exemplu, segmentul de program FOR2 are adresa $25440_{16}$).



Scanned with OKEN Scanner

Adresele modulelor ce compun segmentele programului se stabilesc adăugînd la adresele segmentelor adresele relative de implantare a modulelor în cadrul segmentelor, care sînt indicate în prima situaţie tipărită de editorul de legături (de exemplu, modulul de program COB%%%01, inclus în segmentul de program COB%%%00, are adresa $254C8_{16}+40_{16}=25508_{16}$).

Adresele instrucţiunilor în limbaj-maşină, care sînt incluse în modulele de program, se determină adăugînd la adresele modulelor adresele relative de implantare a acestor instrucţiuni în cadrul modulelor, indicate în lista programului obiect astfel: în prima coloană, în cazul programelor ASSIRIS; în coloana * CPT. ORDI, în cazul programelor COBOL; în prima coloană, partea a treia a listei, în cazul programelor FORTRAN (de exemplu, adresa primei instrucţiuni în limbaj-maşină rezultată prin traducerea instrucţiunii PUT, din linia sursă 13, este $256E8_{16}+154_{16}=2583C_{16}$, $256E8_{16}$ fiind adresa modulului de program A1).

Adresele datelor (variabilelor), care pot fi grupate, în funcţie de limbajul utilizat, în modulele de program, de date sau de date comune, se determină adăugînd la adresele modulelor adresele relative de implantare a acestor date în cadrul modulelor. Aceste adrese relative sînt indicate în geografia datelor astfel: în coloana 4 (VALEUR) în cazul programelor ASSIRIS; în coloana ADRESSE, pentru datele definite în secţiunea WORKING în cazul programelor COBOL (pentru datele definite în secţiunea DATA, adresele acestora se determină adăugînd la adresa modulului de date, adresele relative indicate în coloanele ADRESSE din geografia datelor şi ARCA din geografia fişierelor); în prima coloană a listei variabilelor locale sau a listei variabilelor grupate în blocuri de tip „common" în cazul programelor FORTRAN (de exemplu, adresa variabilei locale F1, definită în programul principal FORTRAN, este $20198_{16}+120_{16}=202B8_{16}$, $20198_{16}$ fiind adresa modulului de program F%MDATA).

Adresele tabelelor de descriere a fişierelor care pot fi incluse, în funcţie de limbajul utilizat, în modulele de program sau de date, se determină adăugînd la adresele modulelor, adresele relative de implantare a acestor tabele în cadrul modulelor. Aceste adrese relative sînt indicate în: geografia datelor, coloana 4, în cazul programelor ASSIRIS; geografia fişierelor, coloana ATDF, însă numai pentru fişierele care nu utilizează unităţile standard ale sistemului, în cazul programelor COBOL (tabelele de descriere a fişierelor care utilizează unităţile standard ale sistemului sînt incluse în modulele TDFSYSIN, TDFSYSUT, TDFSYSPH); prima parte a listei programului obiect (vezi 11.3.1), în cazul programelor FORTRAN (de exemplu, adresa tabelei de descriere a fişierului C2, definit în programul COBOL, este $27698_{16}+168_{16}=27800_{16}$, $27698_{16}$ fiind adresa modulului de date COB%%%00).



Scanned with OKEN Scanner

Adresele primelor zone tampon asociate fişierelor definite în program sînt indicate de cuvintele de adresă relativă $2C_{16}$ din tabelele de descriere a fişierelor (vezi tabelul 11.8). În cazul fişierelor care nu utilizează unităţile standard ale sistemului, definite în programele COBOL, zonele tampon ce le sînt asociate formează segmente de date comune distincte; deci, adresa primei zone tampon asociată unui astfel de fişier coincide cu

*Tabelul 11.8*

**Tabela de descriere a fişierelor**

| Bait | Semnificaţie |
|---|---|
| 0 | Organizarea fişierului : U — nedefinită ; S — secvenţială ; I — secvenţială-indexată ; R — selectivă |
| 1 — 2 | Identificatorul de exploatare al fişierului |
| 12—13 | Numărul de înregistrări fizice citite/scrise din/în fişier |
| 15 | Indicatorul OPEN-CLOSE : $00_{16}$ — fişier închis ; $80_{16}$ — fişier deschis |
| 32—35 | Adresa modulului de acces SGF |
| 36—39 | Adresa blocului de comandă (asociat primei comenzi a programului de canal) |
| 41 | Numărul de zone-tampon asociate fişierului |
| 42—43 | Lungimea maximă a înregistrărilor fizice |
| 44—47 | Adresa primei zone-tampon |
| 48—49 | Lungimea maximă a articolelor |
| 50 | Formatul articolelor, precizat prin valorile ultimelor două cifre binare : 00 — fix ; 01 — variabil ; 10 — nedefinit |
| 51 | Numărul registrului care conţine adresa articolului în curs de prelucrare (în cazul modului de acces LOCATE) |
| 52—54 | Numărul total de căsuţe ale fişierului (în cazul fişierelor selective) |
| 55 | Numărul de căsuţe ale unei pagini (în cazul fişierelor selective) |
| 56—59 | Adresa primei instrucţiuni a secvenţei de sfîrşit de fişier |
| 61 | Lungimea cheii de identificare a articolelor (în cazul fişierelor secvenţial-indexate şi selective) |
| 62 | Numărul registrului care conţine numărul căsuţei (în cazul fişierelor selective) |
| 64—67 | Adresa primei instrucţiuni a secvenţei *cheie eronată* (în cazul fişierelor secvenţia-indexate şi selective) |
| 70—71 | Adresa relativă a cheii în cadrul articolelor (în cazul fişierelor secvenţial-indexate) |

adresa segmentului de date comune respectiv (de exemplu, adresa primei zone tampon a fişierului de idex A2, definit în programul principal FORTRAN, este $23170_{16}$, informaţie furnizată de cuvîntul-memorie de adresă $201F0_{16}$).

Adresele programelor de canal (sau mai exact, a blocurilor de comandă asociate primelor comenzi ale programelor de canal) sînt indicate de cuvintele de adresă relativă $24_{16}$ din tabelele de descriere a fişierelor (de exemplu, pentru acelaşi fişier, adresa primului bloc de comandă este $23140_{16}$, informaţie furnizată de cuvîntul-memorie de adresă $201E8_{16}$).



Scanned with OKEN Scanner

## 11.6. LOCALIZAREA ŞI CORECTAREA ERORILOR SEMNALATE LA EXECUŢIA PROGRAMELOR

O abordare exhaustivă a modalităţilor de localizare şi corectare a tuturor tipurilor de erori care sînt semnalate la execuţie nu este posibilă datorită spaţiului restrîns afectat acestui capitol, cît şi datorită interacţiunii multiplilor factori care pot contribui la producerea acestor erori. De aceea, în cele ce urmează, această abordare va fi de principiu şi va avea în vedere numai erorile de programare.

În majoritatea cazurilor, utilizatorul este informat de componentele sistemului de operare de cauzele erorilor, prin mesaje clare şi explicite, care permit localizarea şi corectarea rapidă a acestora, uneori chiar fără a utiliza instrumente complementare pentru punerea la punct (cum ar fi, de exemplu, lista programului obiect, vidajul memoriei etc.). Există însă şi cazuri cînd localizarea şi corectarea erorilor poate fi laborioasă şi de durată, necesitînd utilizarea acestor instrumente sau chiar revederea logicii programului. Două exemple vor fi date, în continuare, pentru a argumenta ultima afirmaţie.

a) Sistemul de operare informează terminarea anormală a execuţiei unui program, furnizînd mesjaul ERREUR PROGRAMME 00000400 şi vidajul memoriei. Primul cuvînt al registrului de stare program are valoarea $x$, care reprezintă adresa instrucţiunii la execuţia căreia a fost detectată eroarea. Conţinutul cuvîntului-memorie de adresă $x$ este $00000004_{16}$, care indică într-adevăr codul unei operaţii inexistente. Se constată însă că acest cuvînt-memorie reprezintă zona rezervată unei date (variabile).

b) Aceleaşi informaţii furnizate de sistemul de operare. Se constată însă că valoarea $x$ reprezintă adresa unui cuvînt-memorie rezervat unei instrucţiuni.

Întrebările care survin sînt: prin execuţia cărei instrucţiuni de salt s-a transferat controlul execuţiei la adresa $x$? – în primul caz; prin execuţia cărei instrucţiuni a fost modificată instrucţiunea de adresă $x$? – în cel de-al doilea caz.

Cauza care a determinat producerea erorii poate fi intuită: adresare incorectă a operanzilor; în schimb, localizarea erorii este, în general, dificilă, deoarece sistemul, în afară de vidajul memoriei, nu furnizează alte informaţii în acest scop.

Evident că, în ambele din situaţiile prezentate avem erori de programare, întrucît componentele sistemului de operare şi subprogramele din BSS inserate în program (deşi uneori pot prezenta anomalii) nu intervin, în general, în zona de memorie afectată programului.



Scanned with OKEN Scanner

Aspectele evidenţiate pledează pentru o planificare riguroasă a testării şi punerii la punct, care să permită controlul permanent al execuţiei programului. Acest lucru nu poate însă exclude definitiv utilizarea vidajului memoriei ca instrument pentru punerea la punct a programelor, pe care îl considerăm, în unele situaţii, necesar.

În continuare este prezentat modul de abordare a principalelor tipuri de erori semnalate la execuţie, utilizînd ca instrument vidajul memoriei.

11.6.1. **Erori de tip** ERREUR PROGRAMME. Adresa instrucţiunii limbaj-maşină $i$ la execuţia căreia a fost detectată eroarea, pe care o vom nota $A_i$, este indicată de primul cuvînt al registrului de stare a programului. Comparînd adresa $A_i$ cu adresele segmentelor (din tabela segmentelor), pe care le vom nota $A_s$ ($s=1, n$, unde $n$ reprezintă numărul segmentelor programului), se determină segmentul $s$ care conţine instrucţiunea $i$ ($A_s \leqslant A_i < A_{s+1}$). Scăzînd din adresa $A_i$ adresa $A_s$, se determină adresa relativă de implantare a instrucţiunii $i$ în cadrul segmentului $s$, pe care o vom nota cu $A_{rs}$. Comparînd adresa $A_{rs}$ cu adresele relative de implantare în cadrul segmentului $s$ a modulelor care îl compun — adresele relative ale modulelor, pe care le vom nota $A_m$ ($m=1, 2, \ldots, n$), unde $n$ reprezintă numărul modulelor segmentului $s$), sînt indicate în prima situaţie tipărită de editorul de legături — se determină modulul $m$ care conţine instrucţiunea $i$ ($A_m \leqslant A_{rs} < A_{m+1}$).

Dacă modulul $m$ este un modul de program ce nu provine dintr-o bibliotecă de tip BT a utilizatorului sau a sistemului (şi, în general, acesta este cazul obişnuit), scăzînd din adresa $A_{rs}$ adresa $A_m$, se determină adresa relativă de implantare a instrucţiunii $i$ în cadrul modulului $m$, pe care o vom nota $A_{rm}$.

Consultînd lista programului obiect ce corespunde modulului $m$, se poate stabili instrucţiunea în limbaj evoluat $I$, prin traducerea căreia a fost generată instrucţiunea $i$ (în lista programului obiect se caută instrucţiunea în limbaj-maşină a cărei adresă relativă de implantare în cadrul modulului $m$ este egală cu $A_{rm}$).

Corectarea erorii necesită analiza instrucţiunii $I$, a operanzilor citaţi în această instrucţiune, valorile lor, rezultatele execuţiei unor instrucţiuni executate anterior instrucţiunii $I$ etc.

**Exemplu.** Pentru programul 11.1 se consideră următorul mesaj de eroare:

ERREUR PROGRAMME 000000204

şi conţinutul registrului de stare a programului:

ETAT PROGRAMME 00025834 41130000

Instrucţiunea $i$ se află în segmentul de program COB%%%00 (deoarece $254C8_{16} \leqslant 25834_{16} < 27698_{16}$, unde $254C8_{16}$ este adresa segmentului de program COB%%%00,



Scanned with OKEN Scanner

iar $27698_{16}$ adresa segmentului de date COB%%%00), în modulul de program A1 (deoarece $25834_{16}-254C8_{16}=36C_{16}$ și $220 \leqslant 36C < 3B8$, unde $220_{16}$ și $3B8_{16}$ sînt adresele relative de implantare a modulelor A1 și OPCLTYP în cadrul segmentului de program COB%%%00) și are adresa relativă de implantare în cadrul acestui modul $36C_{16}-220_{16}=14C_{16}$, iar instrucțiunea ASSIRIS care îi corespunde este LD4,0 L (24), din linia sursă 11.

Observație. Pentru localizarea erorilor este indicat a utiliza adresele segmentelor din tabela segmentelor și nu cele din registrele de bază, deoarece, uneori, conținutul acestora este eronat.

11.6.2. **Erori de tip** ERREUR E/S. Erorile de programare de acest tip sînt comunicate utilizatorului printr-un mesaj care indică: adresa $A_i+4$ a instrucțiunii care urmează instrucțiunii EXUP sau RST, în cazul erorilor cu codurile 01 și 08, sau adresa blocului de comandă, în cazul celorlalte erori (vezi tabelul 11.6). În primul caz, pentru localizarea erorii, pornind de la adresa instrucțiunii $i$, indicată în mesajul de eroare, se poate stabili modulul și, eventual, instrucțiunea $I$ ce corespunde instrucțiunii $i$ (aceasta numai dacă instrucțiunea $i$ nu figurează într-un modul inserat în program dintr-o bibliotecă de tip BT), iar pentru corectarea erorii trebuie analizat blocul de comandă citat în instrucțiunea $I$. În cel de-al doilea caz, localizarea erorii necesită identificarea unității periferice (care se poate realiza după numărul logic al acesteia, înregistrat în blocul de comandă indicat în mesajul de eroare) și, eventual, a fișierului căruia îi corespunde blocul de comandă.

În general, în cazul programării operațiilor de intrare/ieșire la nivel logic, frecvența de apariție a erorilor de programare de acest tip este mai mică (frecvente fiind, de obicei, erorile cu codurile 10 și 12).

Observație. Fiecărei comenzi a unui program de canal îi este asociat un bloc de comandă de lungime trei (sau patru) cuvinte-memorie care conțin informațiile următoare:

bait 0 — cifrele binare $c_i$ ($i$=0, 2, 4) ale acestui bait au următoarea semnificație:
- $c_0=1$ programul de canal se află în starea „luat în considerare";
- $c_2=1$ s-a produs o eroare de transfer în cursul execuției programului de canal;
- $c_4=1$ în prima locație a zonei tampon a fost transferat caracterul „punct";

bait 4 — numărul logic al unității periferice;

bait 5—7 — adresa primei comenzi a programului de canal;

baiții 8—9 — răspunsuri furnizate în urma execuției instrucțiunilor de intrare/ieșire TDV, AIO sau SIO;

baiții 10—11 — lungimea reziduală (numărul de baiți netransmiși în urma execuției ultimei instrucțiuni de canal).

În cazul în care operațiile de intrare/ieșire se realizează la nivel logic (prin intermediul SGF-ului) adresa primului bloc de comandă al unui program de canal este furnizată de tabela de descriere a fișierelor (cuvîntul de adresă relativă $24_{16}$).



Scanned with OKEN Scanner

**Exemplu.** Fie mesajul de eroare (ipotetic):

01 ERREUR E/S 10024070

Comparînd cuvintele de adresă relativă $24_{16}$, din tabelele de descriere a fişierelor definite în program, cu adresa $24070_{16}$, se găseşte că aceasta reprezintă adresa primului bloc de comandă al programului de canal asociat fişierului pe cartele din programul principal FORTRAN. Cauza erorii poate fi, de exemplu, lipsa cartelei EOF sau a cartelei EOJ. În blocul de comandă de adresă $24070_{16}$ cifra binară $c_4$ a primului bait va avea valoarea 1. Stabilirea exactă a cauzei erorii se poate realiza examinînd conţinutul zonei tampon.

11.6.3. **Erori de tip** DEMANDE INSATISFAITE. Mesajele de interpretare a acestor erori indică adresa $A_i+4$ a instrucţiunii care urmează unei instrucţiuni $i$ de apel al unei funcţii a supervizorului (EXUP, WAIT, RST etc.) pentru care parametrii indicaţi sînt incompleţi sau eronaţi. Stabilirea modulului care conţine instrucţiunea $i$ se realizează conform celor prezentate în 11.6.1. În general, acest tip de eroare apare mai frecvent în cazul apelării funcţiilor supervizorului din programe ASSIRIS (caz în care instrucţiunii $i$ îi poate fi asociată şi instrucţiunea $I$).

**Exemplu.** Mesajul de eroare:

DEMANDE INSATISFAITE 00024A4C

indică faptul că instrucţiunea $i$, de adresă $24A4C_{16}$, este $022E0000_{16}$ şi corespunde unei instrucţiuni EXUP (CSV X'02'). Această instrucţiune se află în modulul de acces SGF asociat fişierului de idex A2, definit în programul principal FORTRAN (adresa modulului de acces SGF — $248B8_{16}$ — este conţinută de cuvîntul-memorie de adresă relativă $20_{16}$ din tabela de descriere a acestui fişier).

11.6.4. **Erori semnalate de** SGF. În general, mesajele de eroare furnizate de SGF sînt destul de explicite, permiţînd localizarea şi corectarea erorilor, prin analiza descrierii fişierelor şi a cartelelor de comandă care le sînt asociate (ASSIGN, FILE, LABEL etc.). Dacă această analiză nu relevă cauza erorii, informaţii suplimentare pot fi obţinute analizînd rezumatul SGF, tabela de descriere a fişierului în cauză, modulul de legătură SGF, zona de reentranţă SGF, conţinutul zonelor tampon şi a zonei articol asociate fişierului, blocurile de comandă etc.

Astfel, în rezumatul SGF — a cărei adresă este indicată de cuvîntul de adresă relativă $4_{16}$ din ZIG — există pentru fişierul respectiv un cuvînt dublu a cărui structură şi semnificaţie este:

baiţii 0—1 — idexul fişierului;

baiţii 4—7 — au valoarea $00000000_{16}$, dacă fişierul nu a fost deschis, sau 01 şi adresa TDF-ului, dacă fişierul a fost deschis.

În tabela de descriere a fişierului, baiţii de adresă relativă $E_{16}$ şi $F_{16}$ indică starea fişierului (vezi tabelul 11.8), iar semicuvîntul de adresă relativă $C_{16}$ numărul de înregistrări fizice citite/scrise.



Scanned with OKEN Scanner

În modulul de legătură SGF (a cărui adresă este indicată de primul cuvînt al ZIG), cuvîntul de adresă relativă $1C_{16}$ indică adresa de revenire $A_i+4$, în programul utilizator, după execuţia ultimei funcţii OPEN-CLOSE. De asemenea, în zona de reentranţă SGF (a cărei adresă este indicată de cuvîntul de adresă relativă $40_{16}$ din ZIG), cuvîntul de adresă relativă $4_{16}$ indică adresa de revenire $A_i+4$, în programul utilizator, după execuţia ultimei operaţii de intrare/ieşire. Pe baza adresei $A_i+4$ pot fi stabilite instrucţiunile $i$ (şi corespondentele lor, $I$) care au apelat funcţiile SGF.

Informaţii utile pentru localizarea şi corectarea erorilor SGF poate furniza şi programul de inventariere a fişierelor, prin cele două funcţii ale sale: inventarierea fişierelor şi rezumarea volumelor. Identificarea fişierului la prelucrarea căruia a fost semnalată o eroare SGF se realizează după identificatorul de exploatare indicat în mesajul de eroare.

**Exemplu.** Mesajul de eroare:

C1EGFOE 29 F.NON TROUVE

arată că fişierul de idex C1 (fişierul A3, din programul ASSIRIS) nu este găsit pe suportul fizic (volumul montat pe unitatea de discuri RD1).

11.6.5. **Erori semnalate de subprogramele din** BSS. Mesajele de interpretare a acestor erori (vezi 11.4) indică adresa instrucţiunii $i$ care a apelat subprogramul. Identificarea instrucţiunii $i$ ce corespunde instrucţiunii $i$ se realizează conform celor prezentate în 11.6.1.

**Exemplu.** Mesajul de eroare:

ERREUR : FORMAT (ADRESSE : 020368) CARACTERE ILLEGAL
DANS LA CHAINE D'ENTREE

indică faptul că instrucţiunea $i$ se află în segmentul de program FOR1 ($20128_{16} \leqslant \leqslant 20638_{16} < 24FB0_{16}$), în modulul F%MDATA ($70_{16} \leqslant 20368_{16} - 20128_{16} < 328_{16}$), are adresa relativă de implantare în cadrul acestui modul $1D0_{16}$ şi a fost generată prin traducerea instrucţiunii READ, din linia sursă 7 a programului principal FORTRAN.

Scanned with OKEN Scanner

# BIBLIOGRAFIE


1. Alfonseca, M., Tavera, M. L., Casajuana, R. *An APL Interpreter and System for a Small Computer.* În: IBM System Journal, **16**, nr. 1, ian., 1977, p. 18—40.
2. Amstrong, R. M. *Modular Programming in COBOL.* New York, John Wiley, 1973.
3. Aron, J. D. *The Program Development Process; The Individual Programmer.* Addison—Wesley, Reading, Mass., 1974.
4. Assabgui, M. *Le language d'assemblage; L'assembleur OS/360.* Paris, Dunod, 1972.
5. Baltag, V. *Sistemul FELIX C-256; Structura și programarea calculatorului.* București, Editura tehnică, 1974.
6. Backus, J. W. *The Syntax and Semantics of the Proposed International Algebric Language of Zurich ACM-GAMM Conference.* În: Proc. International Conf. Information Processing UNESCO, Paris, iunie 1959, p. 125—132.
7. Berthel, Ch. *Le Language de programmation PL/I.* Paris, Dunod, 1971.
8. Berztiss, A. T. *Data Structures; Theory and Practice.* New York, Academic Press, 1971.
9. Bauer, F. L. *Software Engineering.* Berlin, Springer Verlag, 1973, p. 279—318.
10. Bobrow, D. G. *Storage management in LISP.* În: Symbol Manipulation Languages and Techniques. Bobrow, D.W. (ed.), Amsterdam, North-Holland, 1968.
11. Bohm, C., Jacopini, G. *Flow Diagrams. Turing Machines and Languages with Only Two Formation Rules.* În: Communications of ACM, **9**, nr. 5, mai 1966, p. 366—371.
12. Brown, A. R., Sampson, W. A. *Program Debugging.* London, Mac Donald, 1973.
13. McCarthy, J. *Recursive Functions of Symbolic Expressions and Their Computation by Machine.* În: Communications of ACM, **3**, nr. 4, aprilie 1960, p. 184—195.
14. Church, A. *The Calculi of Lambda Conversion.* Princeton, N. J., Princeton University Press, 1941.
15. McCracken, D. C. *A Simplified Guide to Structured COBOL Programming* Inc., John Wiley, 1976.
16. Dahl, O. J., Dijkstra, E. W., Hoare, C. A. R. *Structured Programming.* London, New York, Academic Press, 1972.
17. Dimo, P. *Programarea în FORTRAN.* București, Editura didactică și pedagogică, 1972.
18. Dijkstra, E. W. *A Discipline of Programming.* Englewood Cliffs, New Jersey. Prentice Hall, Inc., 1976.
19. Drăghici, M. *Inițiere în COBOL.* București, Editura tehnică, 1972.
20. Dupont, Ph., Tallineau, Y. *Les languages de programmation en parallèle: FORTRAN, COBOL, PL/I, APL.* Paris, Masson et Cie Editeurs, 1974.




Scanned with OKEN Scanner

21. Farber, D. J., Griswold, R. E., Polonsky, I. P. *SBNOBOL, A String Manipulation Language.* În : Journal of ACM, **11**, ianuarie 1964, p. 21.
22. Gelereter, H., Hansen, J. R., Gerberich, C. L. *A FORTRAN—Compiled List-Processing Language.* În : Journal of ACM, **7**, aprilie 1960.
23. Georgescu, H. *Programarea în limbajul PL/I.* București, Editura Academiei, 1973.
24. Gordon, G. A. *A General System Simulator.* În : IBM System Journal, **F**, 1962.
25. McGowan, C. L., Kelly, J. R. *Top-Down Structured Programming Thechniques.* New York, Petrocelly/Charter, 1975.
26. Griswold, R. E. *String and List Proccessing în SNOBOL4.* Englewood Cliffs, New Jersey, Prentice Hall, Inc., 1975.
27. Hansen, P. B. *Concurent Programming Concepts.* În : ACM Computing Surveys, **5**, nr. 4, dec. 1973, p. 223—245.
28. Hassitt, A., Lyon, L. E. *An APL Emulator on System/370.* În : IBM System Journal, **15**, nr. 4, 1976, p. 358—378.
29. Hetzel, W. C. *Program Test Methods.* Englewood Cliffs, N.J., Prentice Hall. Inc., 1973.
30. Hoare, C. A. R. *Notes on Data Structuring.* In Structured Programming de Dahl, O.J., Dijkstra, E. W., Hoare, C. A. R. London, New York, Academic Press, 1972, p. 83—174.
31. Iverson, K. E., *A Programming Language.* New York, John Wiley, 1962.
32. Jitaru, M., Teodorescu, Al. *Programarea la calculatoarele electronice ; limbajul ASSIRIS.* București, Editura didactică și pedagogică, 1975.
33. Knuth, D. E. *Sortare și căutare* (trad. din l. engleză). București, Editura tehnică, 1976.
34. Knuth, D. E. *The Art of Computer Programming.* Vol. I, Mass., Addison-Wesley, Reading, 1968.
35. Lapidus, A., Goldstein, M. *Some Experiments în Algebric Manipulation by Computer.* În : Communications of ACM, august 1965, p. 501.
36. Leavenworth, B. M., Sammet, K. E. *An Overview of Nonprocedural Languages.* În : SIGPLAN Notices, **9**, nr. 4, aprilie 1974, p. 1—12.
37. Ledgard, H. F. *A Model for Type Checking and Application to ALGOL 60.* În : Communication of ACM, **15**, nr. 11, noiembrie 1972, p. 956—966.
38. Ledgard, H. F., Marcotty, M. *A Genealogy of Control Structures.* În : Communications of ACM, **18**, nr. 11, noiembrie 1975, p. 629—639.
39. Lee, R. C. T., Chang, S. K. *Structured Programming and Automatic Program Synthesis.* În : SIGPLAN Notices, **6**, nr. 4, aprilie 1974, p. 60—70.
40. Lientz, B. P. *A Comparative Evaluation of Versions of BASIC.* În : Communications of ACM, **19**, nr. 4, aprilie 1976, p. 175—181.
41. Lovin, M., Preoteasa, P., Popovici, C. P. *Informatica.* București, Editura didactică și pedagogică, 1973.
42. Macarie, C., Pitiș, A. *Limbaje de programare -COBOL.* București, Editura didactică și pedagogică, 1976.
43. Maynard, J. *Programmation modulaire.* Paris. Masson et Cie, 1973.
44. Miller, R. E. *A Comparison of Some Theoretical Models of Parallel Computation.* În : IEEE Trans. on Computers, **C-22**, nr. 8, august 1973, p. 710—717.
45. Mircescu, C., Teodorescu, Al. Jitaru, M., Diatcu, E. *Sisteme de prelucrare automată a datelor ; limbaj ASSIRIS.* București, Editura didactică și pedagogică, 1976.
46. Munteanu, E. *Limbajul ASSIRIS.* București, Editura tehnică, 1976.
47. Morris, C. *Fundation of the theory of Signs.* În : International Encyclopedia of Unified Science, vol. I, nr. 2, Chicago, 1938.



Scanned with OKEN Scanner